

Stanford CS224W: Community Detection in Networks

CS224W: Machine Learning with Graphs

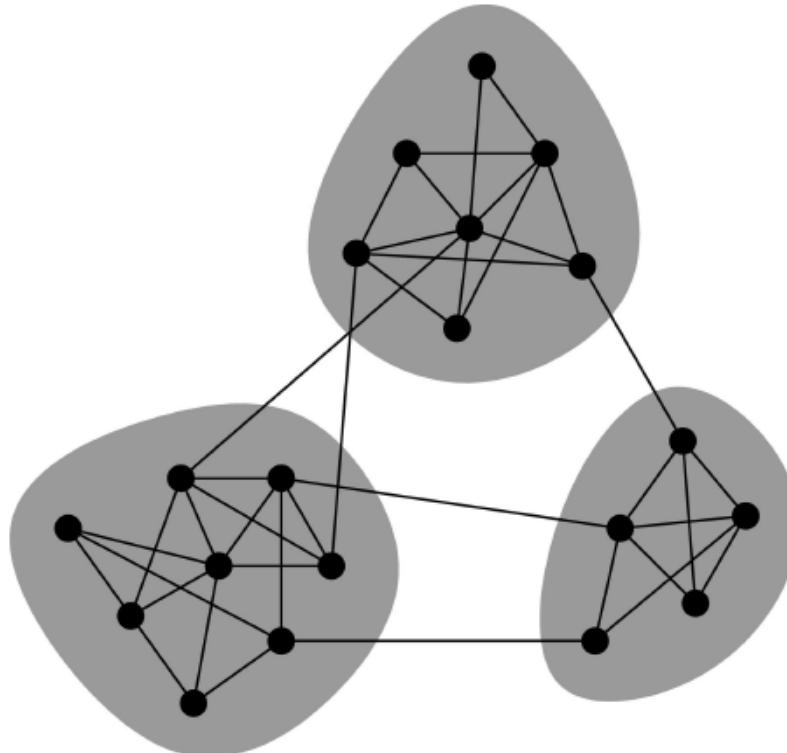
Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>



Networks & Communities

- We often think of networks “looking” like this:

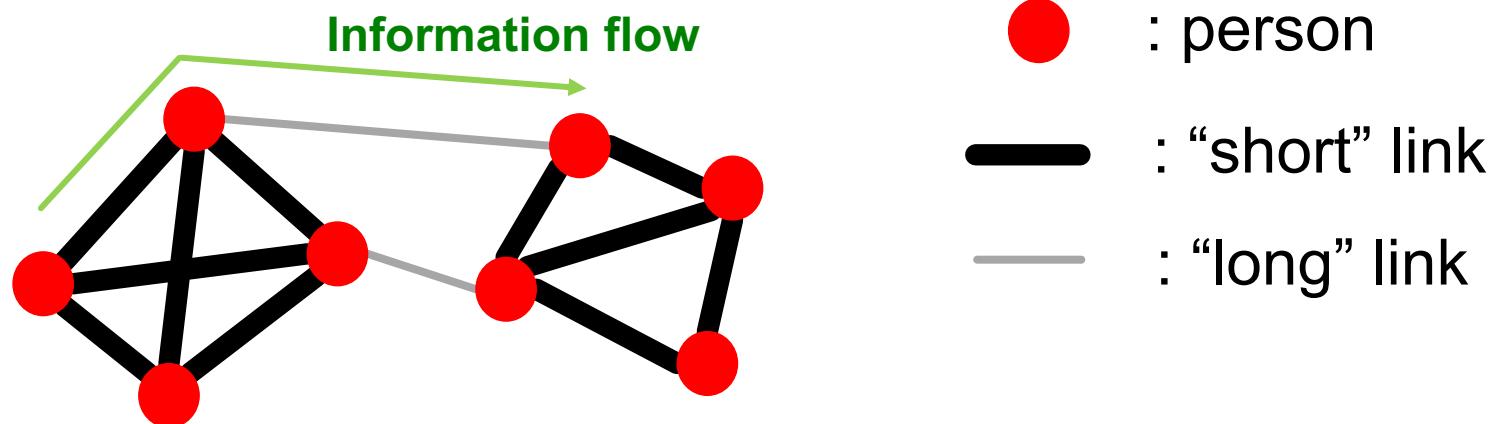


- What led to such a conceptual picture?

Networks: Flow of Information

- How does information flow through the network?
 - People are “embedded” in a social network.
 - There are different links (“short” vs. “long”) in the network, through which information flows.

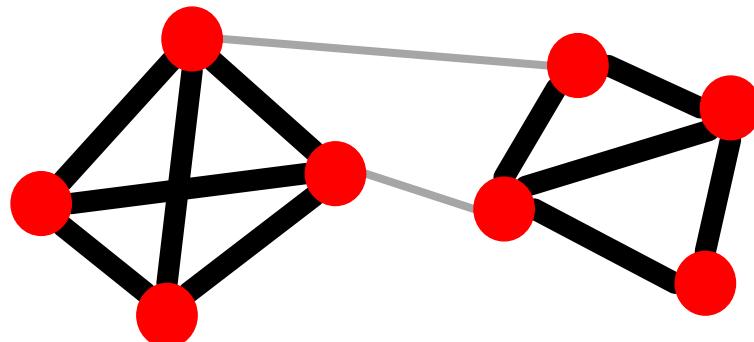
Social network



Flow of Job Information

- **How do people find out about new jobs?**
 - Mark Granovetter, part of his PhD in 1960s
 - People find the information **through personal contacts**
- **But:** Contacts were often **acquaintances** rather than close friends
 - **This is surprising:** One would expect your friends to help you out more than casual acquaintances
- **Why is it that acquaintances are most helpful?**

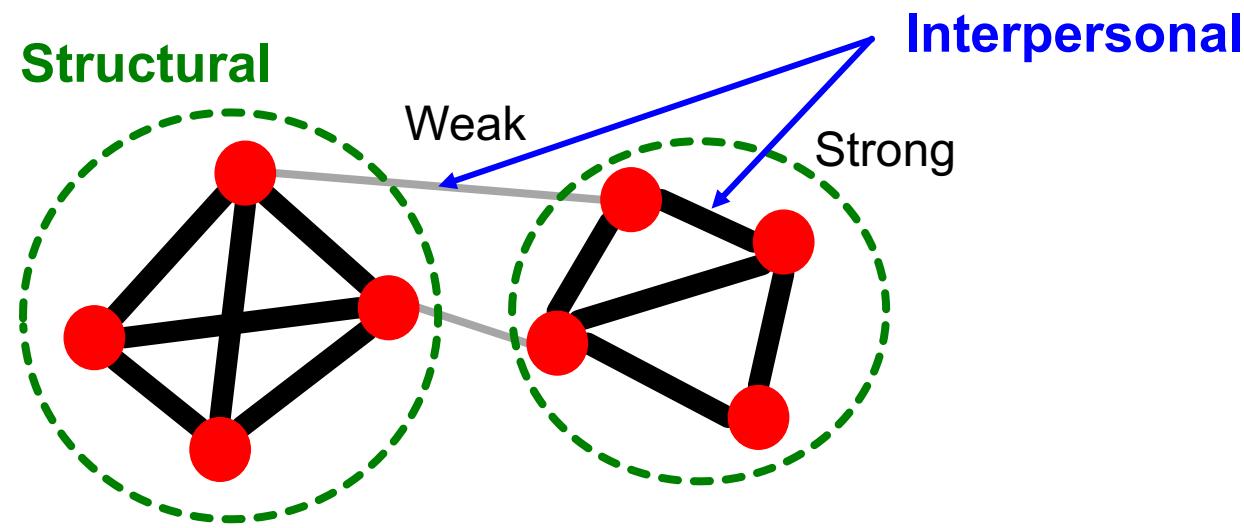
Personal contact network



- : “close-friend” link
- : “acquaintance” link

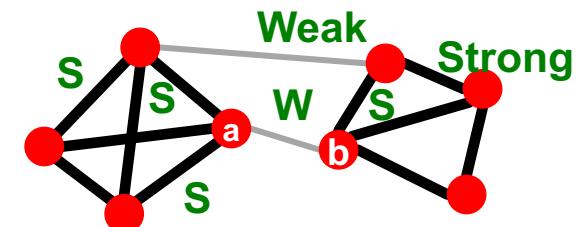
Granovetter's Answer

- Two perspectives on **friendships**:
 - **Structural**: Friendships span different parts of the network
 - **Interpersonal**: Friendship between two people is either **strong** or **weak**



Granovetter's Explanation

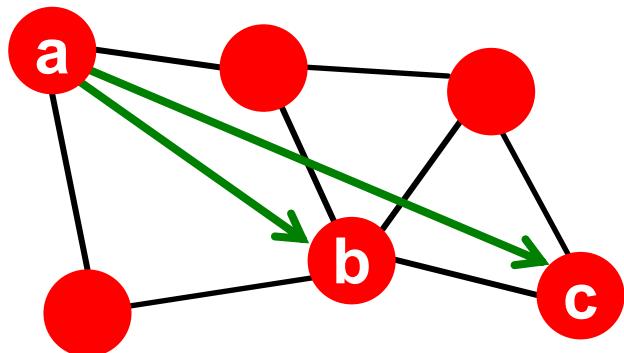
- Granovetter makes a connection between the social and structural role of an edge
- First point: Structure
 - Structurally embedded (tightly-connected) edges are also socially **strong**
 - Long-range edges spanning different parts of the network are socially **weak**
- Second point: Information
 - Long-range edges allow you to gather information from different parts of the network and get a job
 - Structurally embedded edges are heavily redundant in terms of information access



Triadic Closure

- How community (tightly-connected cluster of nodes) forms?

Which edge is more likely, a-b or a-c?



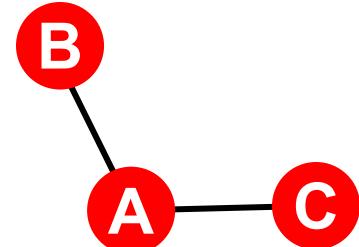
If two people in a network have a friend in common, then there is an increased likelihood they will become friends themselves.

Reasons for Triadic Closure

- **Triadic closure = High clustering coefficient**

Reasons for triadic closure:

- If **B** and **C** have a friend **A** in common, then:
 - **B** is more likely to meet **C**
 - (since they both spend time with **A**)
 - **B** and **C** trust each other
 - (since they have a friend in common)
 - **A** has **incentive** to bring **B** and **C** together
 - (since it is hard for **A** to maintain two disjoint relationships)
- **Empirical study by Bearman and Moody:**
 - Teenage girls with low clustering coefficient are more likely to contemplate suicide



Edge Strength in Real Data

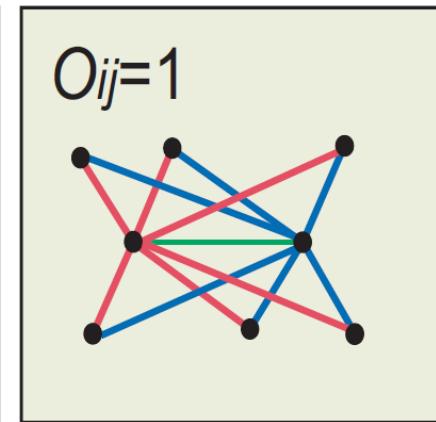
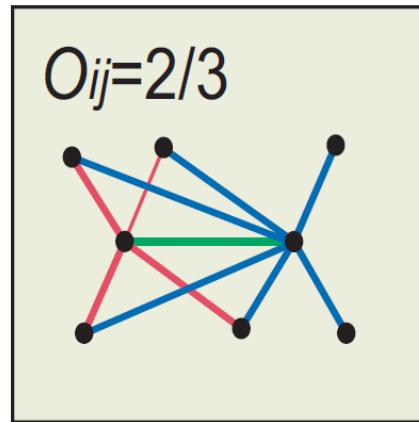
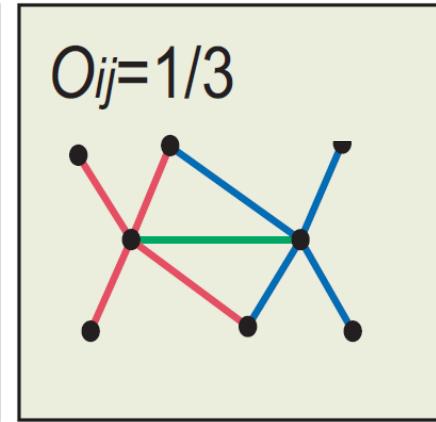
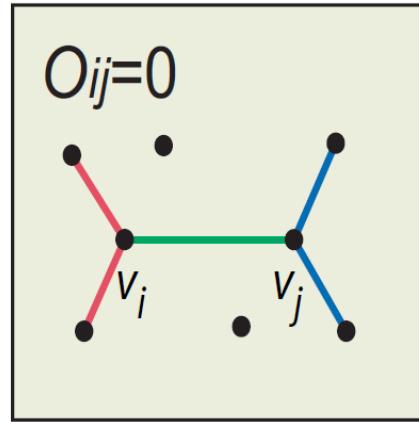
- For many years Granovetter's theory was not tested
- But, today we have large who-talks-to-whom graphs:
 - Email, Messenger, Cell phones, Facebook
- Onnela et al. 2007:
 - Cell-phone network of 20% of EU country's population
 - Edge weight: # phone calls

Edge Overlap

■ Edge overlap:

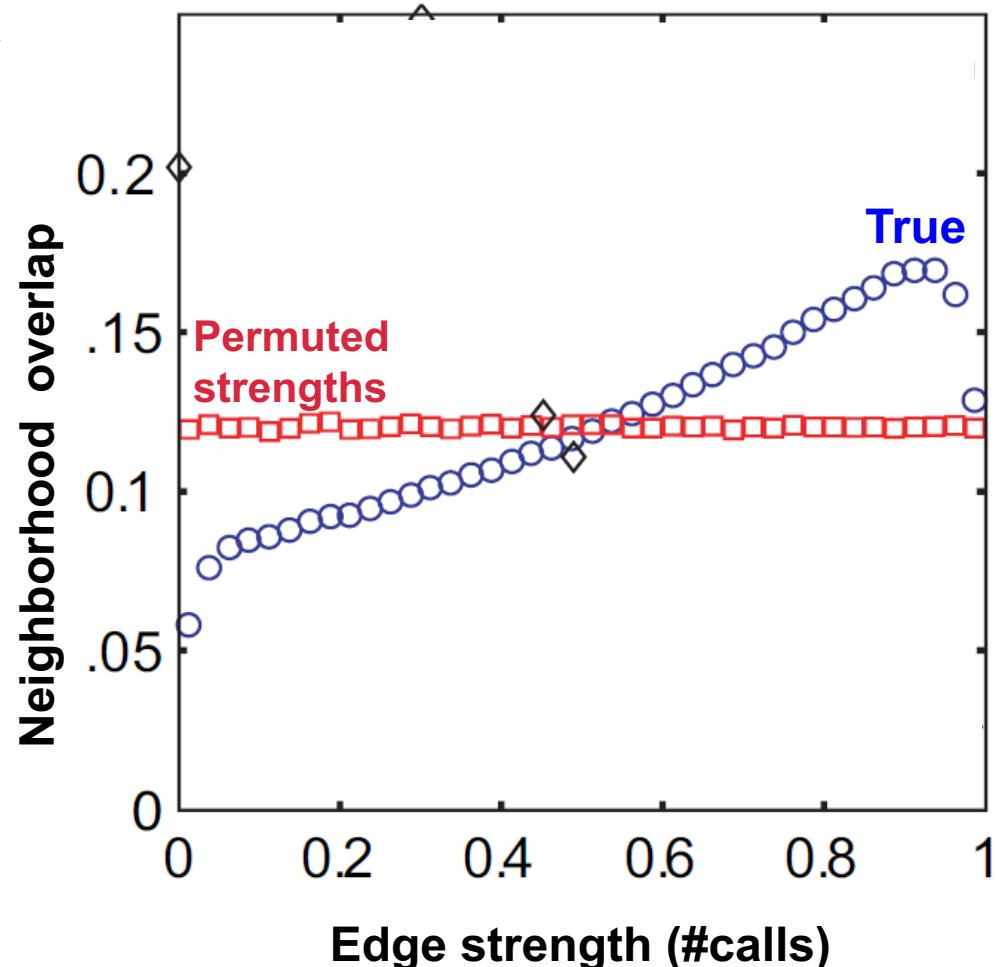
$$O_{ij} = \frac{|(N(i) \cap N(j)) - \{i, j\}|}{|(N(i) \cup N(j)) - \{i, j\}|}$$

- $N(i)$... the set of neighbors of node i
- Note: Overlap = 0 when an edge is a **local bridge**

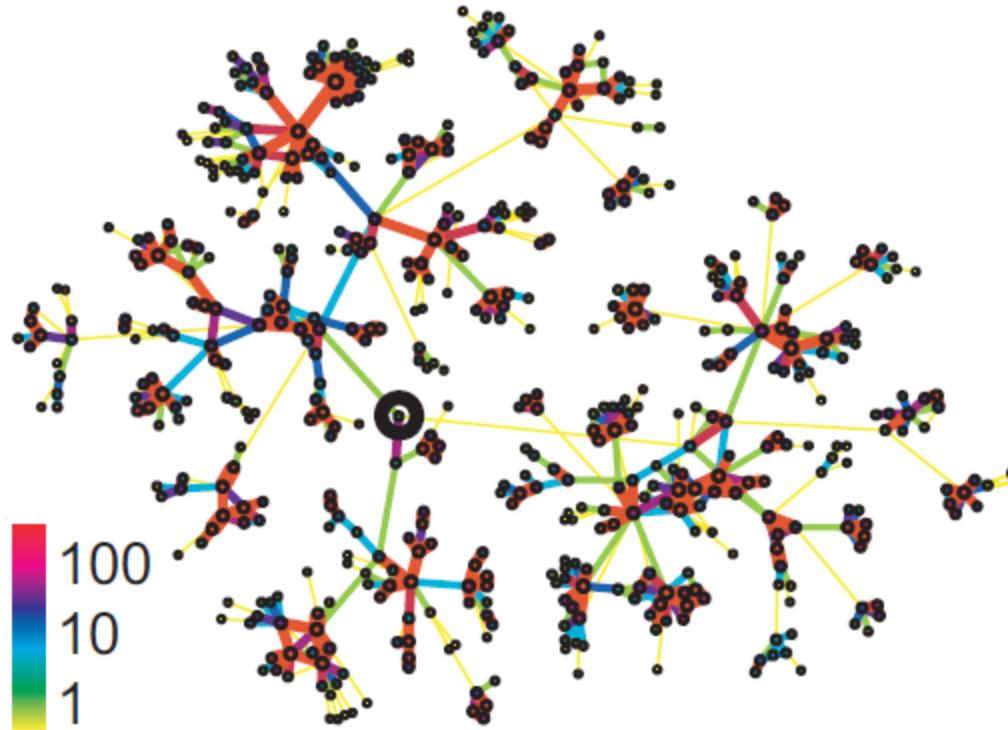


Phones: Edge Overlap vs. Strength

- Cell-phone network
- Observation:
 - Highly used links have high overlap!
- Legend:
 - True: The data
 - Permutated strengths: Keep the network structure but randomly reassign edge strengths

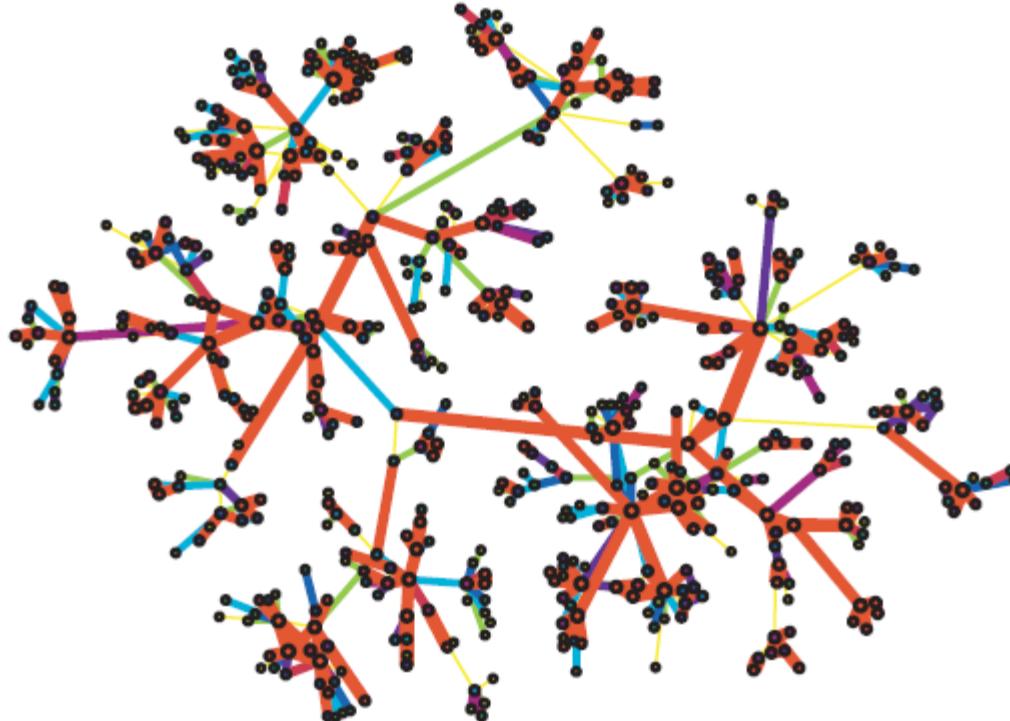


Real Network, Real Edge Strengths



- **Real edge strengths in mobile call graph**
 - Strong ties are more embedded (have higher overlap)

Real Net, Permuted Tie Strengths

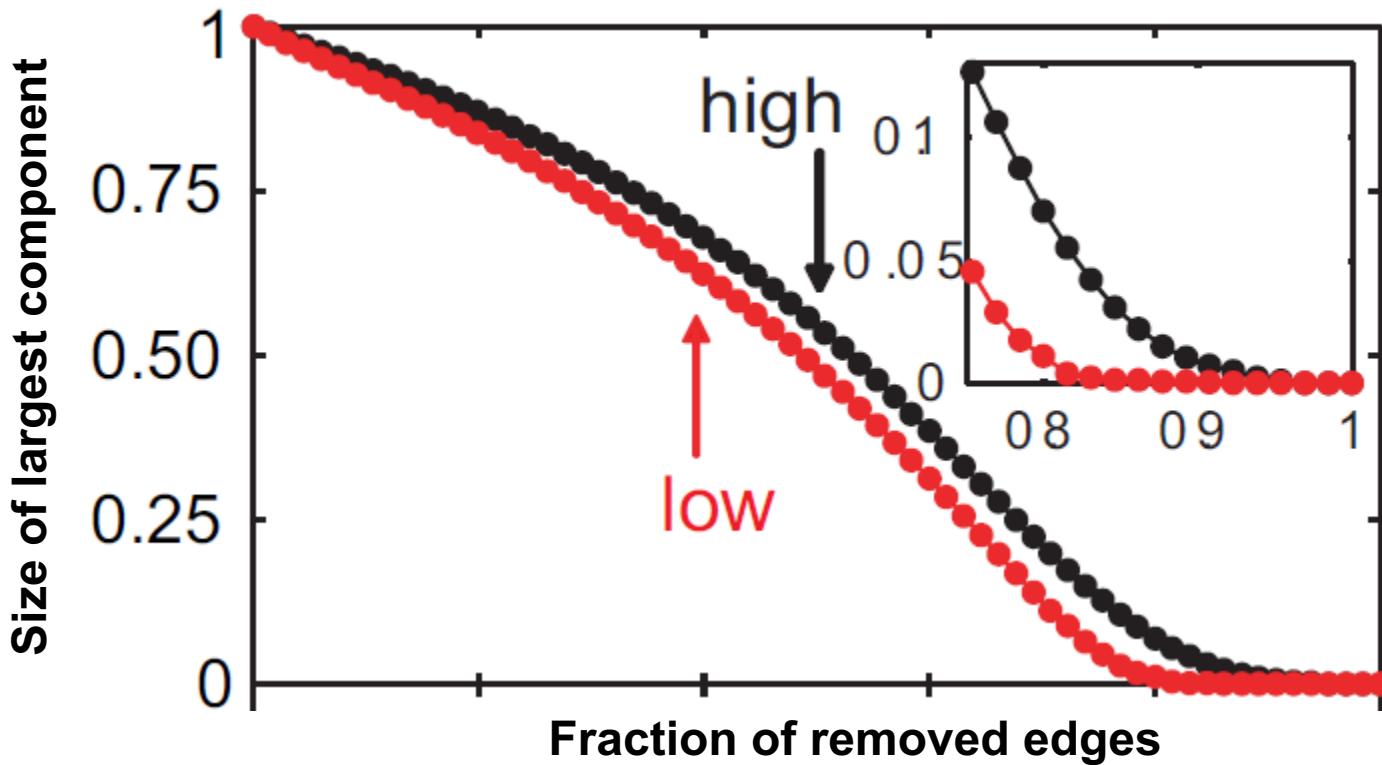


- Same network, same set of edge strengths
but now **strengths are randomly shuffled**

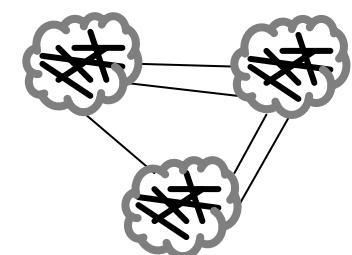
Edge Removal by Strength

Removing edges based on **strength (#calls)**

- Low to high
- High to low



Low
disconnects
the network
sooner

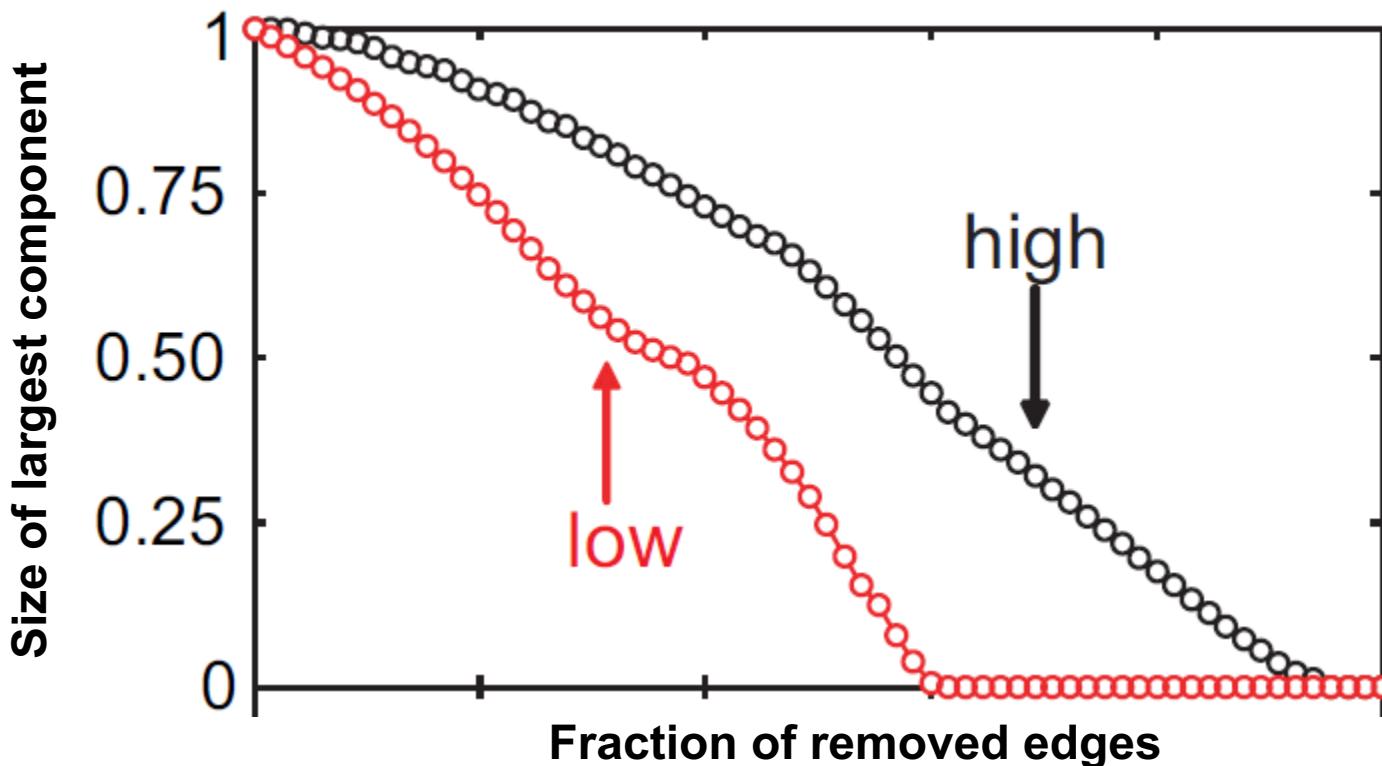


Conceptual picture
of network structure

Edge Removal by Overlap

Removing edges based on **edge overlap**

- Low to high
- High to low

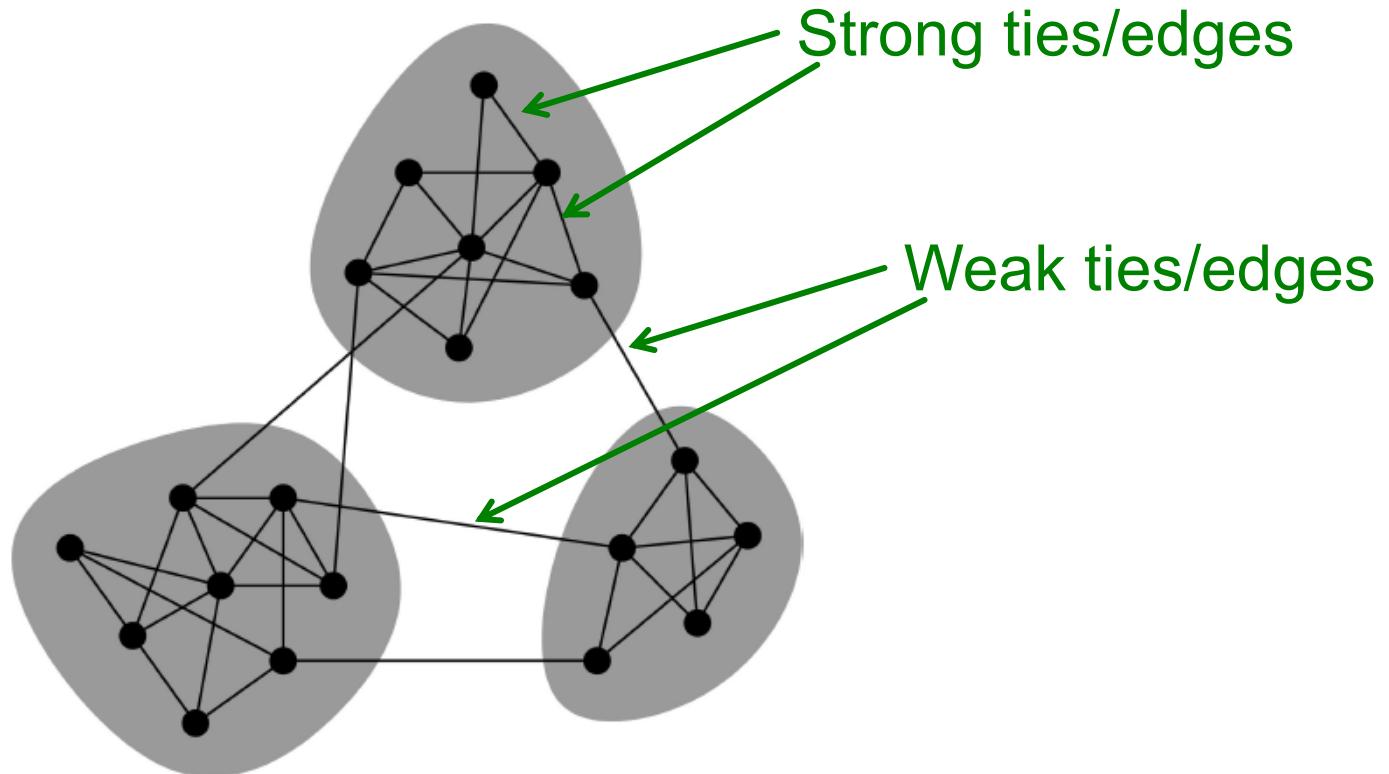


Low
disconnects
the network
sooner

Conceptual picture
of network structure

Conceptual Picture of Networks

- Granovetter's theory leads to the following conceptual picture of networks



Stanford CS224W: **Network Communities**

CS224W: Machine Learning with Graphs

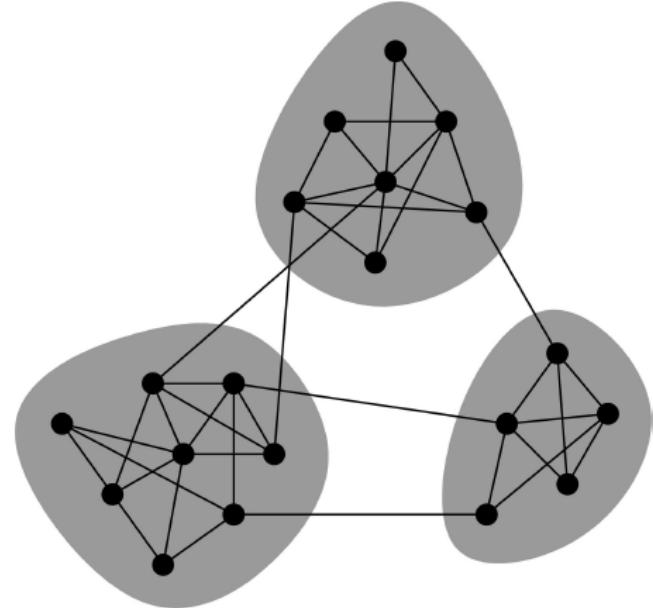
Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>



Network Communities

- Granovetter's theory suggests that networks are composed of **tightly connected sets of nodes**



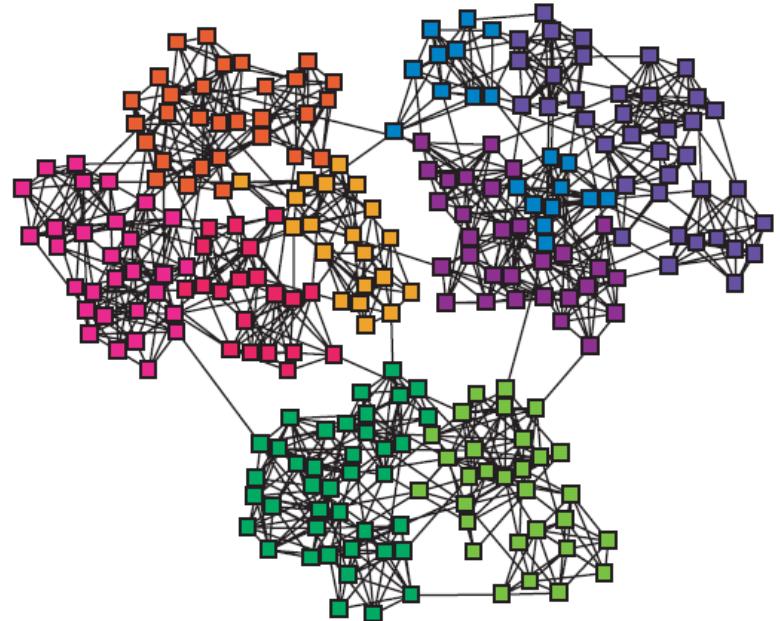
Communities, clusters,
groups, modules

- **Network communities:**

- Sets of nodes with **lots of internal** connections and **few external** ones (to the rest of the network).

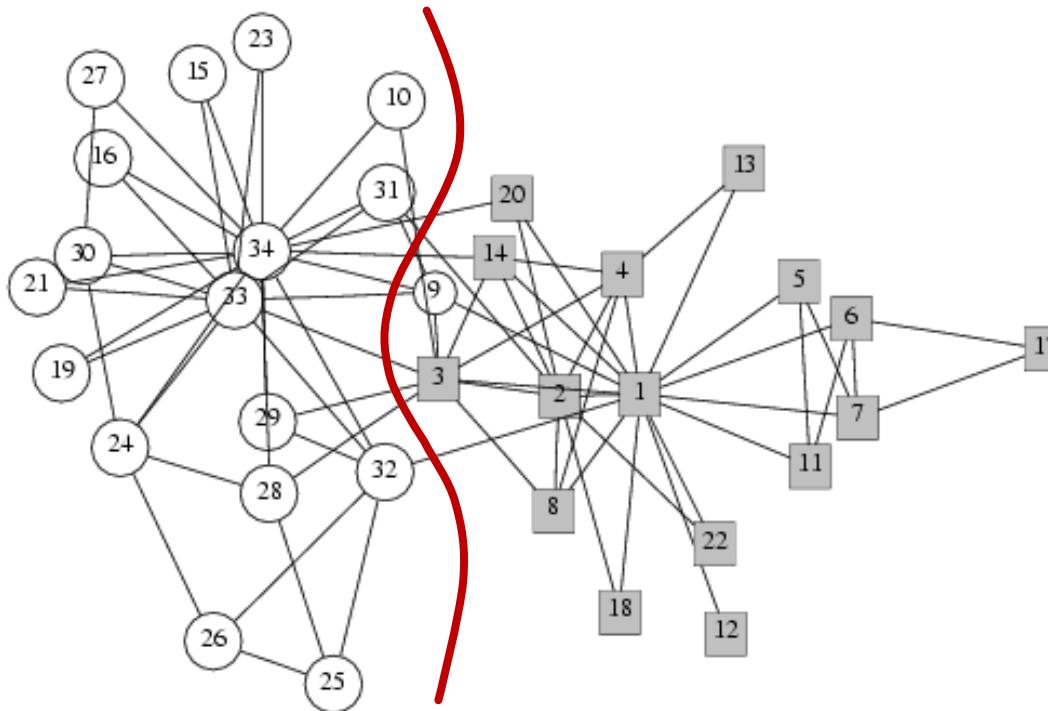
Finding Network Communities

- **How do we automatically find such densely connected groups of nodes?**
- Ideally such automatically detected clusters would then correspond to real groups
- **For example:**



Communities, clusters,
groups, modules

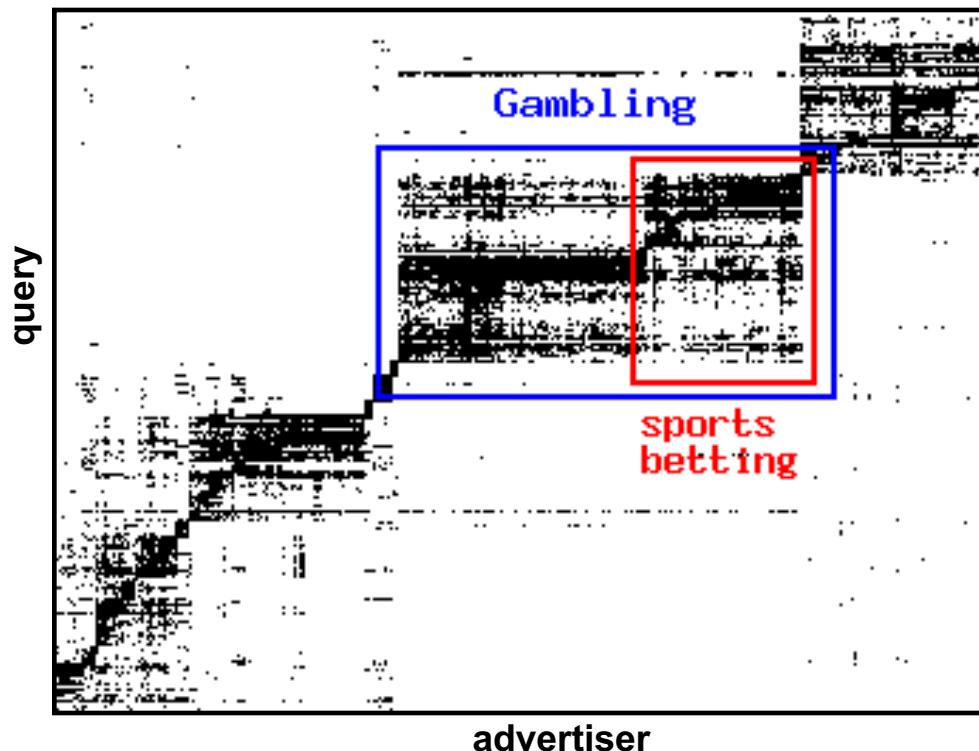
Social Network Data



- **Zachary's Karate club network:**
 - Observed social ties & rivalries in a university karate club
 - During the study, conflicts led the group to split
 - Split could be explained by a minimum cut in the network

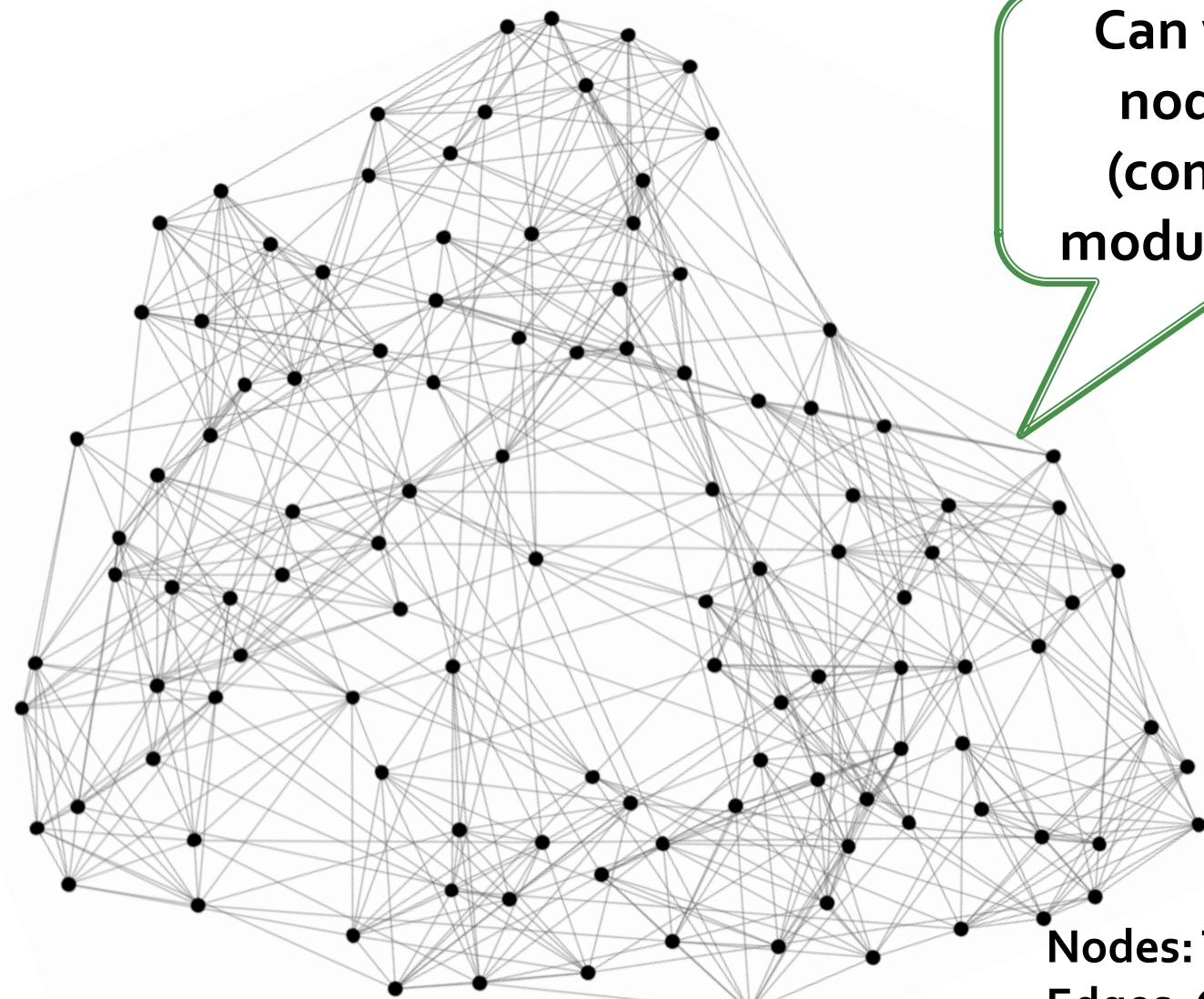
Micro-Markets in Sponsored Search

Find micro-markets by partitioning the “query-to-advertiser” graph in web search:



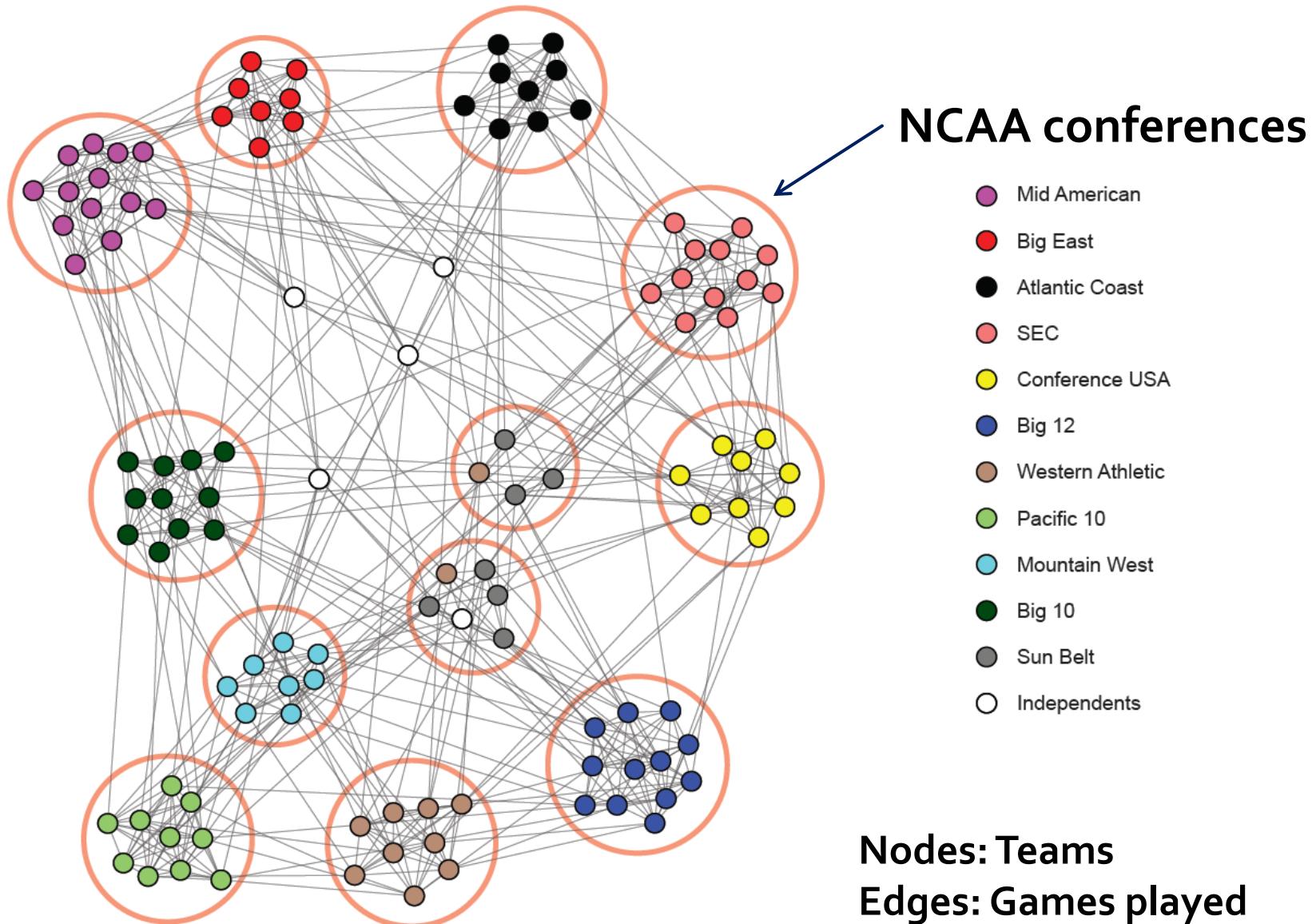
Nodes: advertisers and queries/keywords; Edges: Advertiser advertising on a keyword.

NCAA Football Network



Can we identify
node groups?
(communities,
modules, clusters)

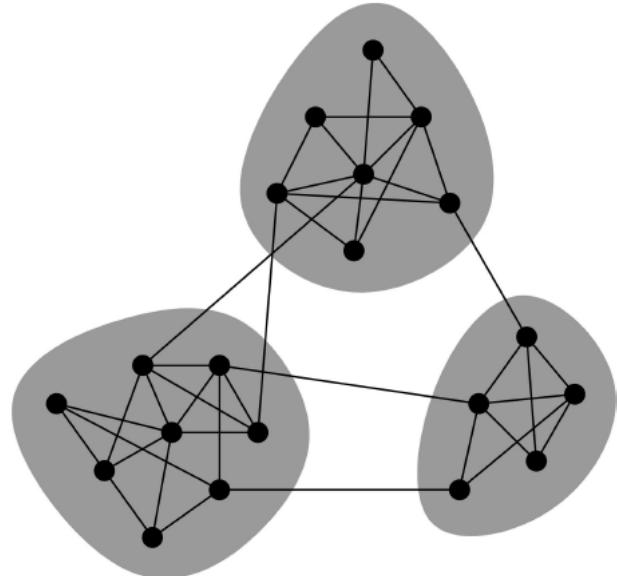
NCAA Football Network



Network Communities

- **Communities:** sets of **tightly connected nodes**
- Define: **Modularity Q**
 - A measure of how well a network is partitioned into communities
 - Given a **partitioning** of the network into groups disjoint $s \in S$:

$$Q \propto \sum_{s \in S} [(\# \text{ edges within group } s) - \underbrace{(\text{expected } \# \text{ edges within group } s)}_{\text{Need a null model}}]$$

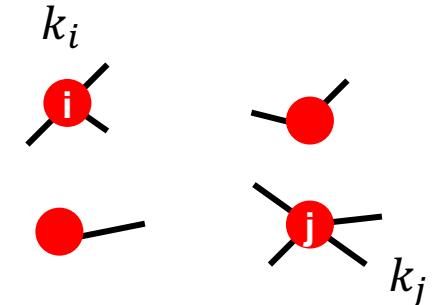


Null Model: Configuration Model

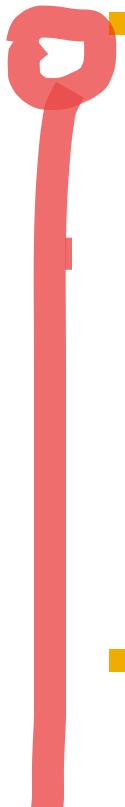
- Given real G on n nodes and m edges, construct rewired network G'

- Same degree distribution but uniformly random connections
- Consider G' as a multigraph (multiple edges exist between nodes)

- The expected number of edges between nodes i and j of degrees k_i and k_j equals: $k_i \cdot \frac{k_j}{2m} = \frac{k_i k_j}{2m}$
 - There are 2m directed edges (counting $i \rightarrow j$ and $j \rightarrow i$) in total.
 - For each of k_i out-going edges from node i , the chance of it landing to node j is $k_j/2m$, hence $k_i k_j/2m$.



Null Model: Configuration Model



- The expected number of edges between nodes i and j of degrees k_i and k_j equals: $\mathbf{k}_i \cdot \frac{\mathbf{k}_j}{2m} = \frac{\mathbf{k}_i \mathbf{k}_j}{2m}$
 - The expected number of edges in (multigraph) \mathbf{G}' :
 - $= \frac{1}{2} \sum_{i \in N} \sum_{j \in N} \frac{k_i k_j}{2m} = \frac{1}{2} \cdot \frac{1}{2m} \sum_{i \in N} k_i (\sum_{j \in N} k_j) =$
 - $= \frac{1}{4m} 2m \cdot 2m = m$
- Under null model, both the degree distribution and the total number of edges are preserved.

Note:
$$\sum_{u \in N} k_u = 2m$$

In directed graphs, $\text{sum}(\text{vertex_degree}) = 2 * \text{sum}(\text{edge})$

Notice: This model applies to both weighted and unweighted networks. For weighted networks we use the weighted degree (sum of the edge weights).

Modularity

- **Modularity of partitioning S of graph G :**
 - $Q \propto \sum_{s \in S} [(\# \text{ edges within group } s) - (\text{expected } \# \text{ edges within group } s)]$
 - $$Q(G, S) = \frac{1}{2m} \underbrace{\sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)}$$

Normalizing const.: $-1 \leq Q \leq 1$
- **Modularity values take range $[-1, 1]$**
 - It is positive if the number of edges within groups exceeds the expected number
 - Q greater than 0.3-0.7 means significant community structure
 - Notice Modularity applies to weighted and unweighted networks.

RECAP: Modularity

For each group s

$$Q(G, S) = \frac{1}{2m} \sum_{s \in S} \left(\sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \right)$$

Equivalently modularity can be written as:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j)$$

- A_{ij} represents the edge weight between nodes i and j ;
- k_i and k_j are the sum of the weights of the edges attached to nodes i and j , respectively;
- $2m$ is the sum of all of the edge weights in the graph;
- c_i and c_j are the communities of the nodes; and
- δ is an indicator function $\delta(c_i, c_j) = 1$ if $c_i = c_j$ else 0

Idea: We can identify communities by maximizing modularity

Stanford CS224W: Louvain Algorithm

CS224W: Machine Learning with Graphs

Jure Leskovec, Stanford University

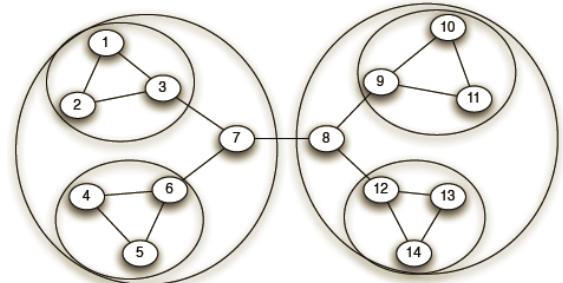
<http://cs224w.stanford.edu>



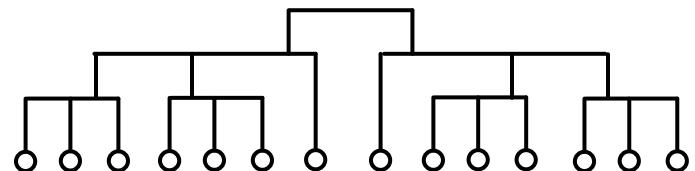
Louvain Algorithm

- **Greedy algorithm** for community detection
 - $O(n \log n)$ run time
- Supports weighted graphs
- Provides hierarchical communities
- Widely utilized to **study large networks** because:
 - Fast
 - Rapid convergence
 - High modularity output
(i.e., “better communities”)

Network and communities:



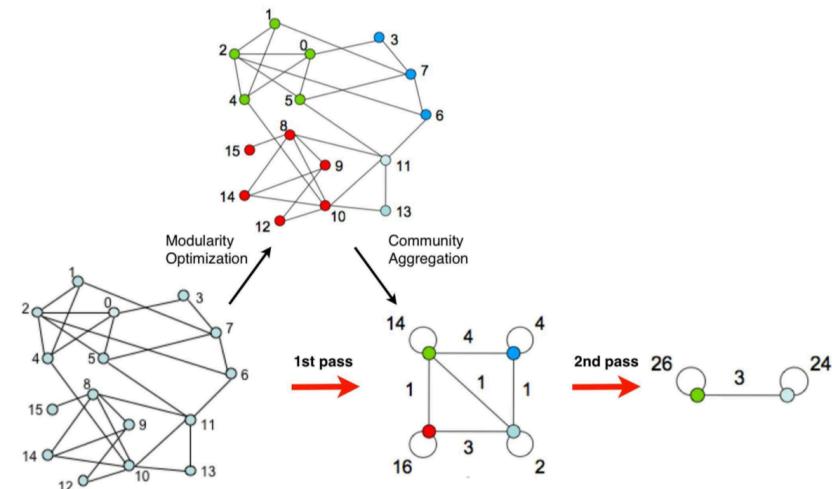
Dendrogram:



Louvain Algorithm: At High Level

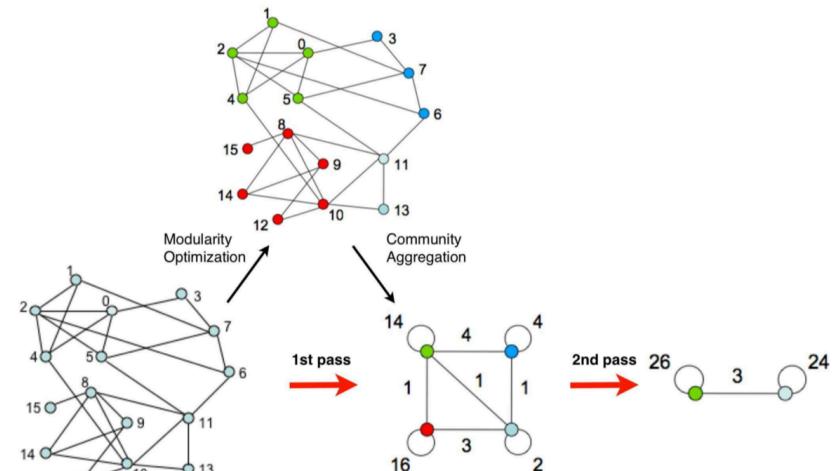
- Louvain algorithm **greedily maximizes** modularity
- **Each pass is made of 2 phases:**
 - **Phase 1:** Modularity is **optimized** by allowing only local changes to node-communities memberships
 - **Phase 2:** The identified communities are **aggregated** into super-nodes to build a new network
 - **Goto Phase 1**

The passes are repeated **iteratively** until no increase of modularity is possible.



Louvain Algorithm: At High Level

- Louvain algorithm considers graphs as **weighted**
 - The original graph can be unweighted (i.e., edge weights are all 1)
 - As the communities get identified and aggregated into super-nodes, weighted graphs are created (weights count the number of edges in the original graph)
- Weighted version of Modularity is applied



Louvain: 1st phase (Partitioning)

- 
- Put each node in a graph into a **distinct community** (one node per community)
 - For each node i , the algorithm performs two calculations:
 - Compute the modularity delta (ΔQ) when putting node i into the community of some neighbor j
 - Move i to a community of node j that yields the largest gain in ΔQ
 - **Phase 1 runs until no movement yields a gain**

This first phase stops when a local maxima of the modularity is attained, i.e., when no individual node move can improve the modularity.

Note that the output of the algorithm depends on the order in which the nodes are considered.

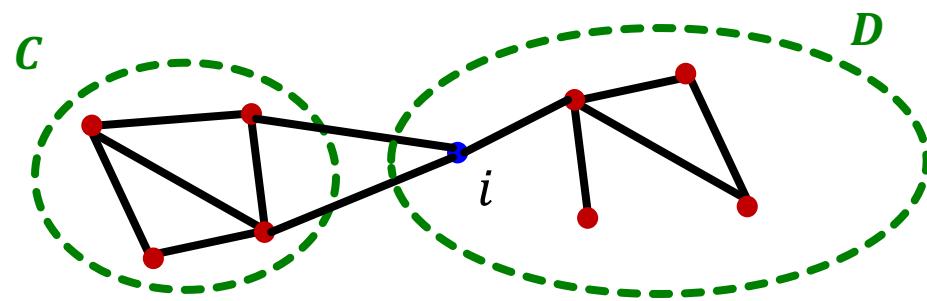
Research indicates that the ordering of the nodes does not have a significant influence on the overall modularity that is obtained.

Louvain: Modularity Gain

What is ΔQ if we move node i to community D to C ?

$$\Delta Q(D \rightarrow i \rightarrow C) = \Delta Q(D \rightarrow i) + \Delta Q(i \rightarrow C)$$

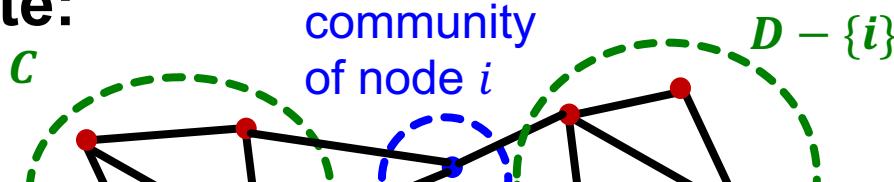
Before:



Removing i from D

$$\Delta Q(D \rightarrow i)$$

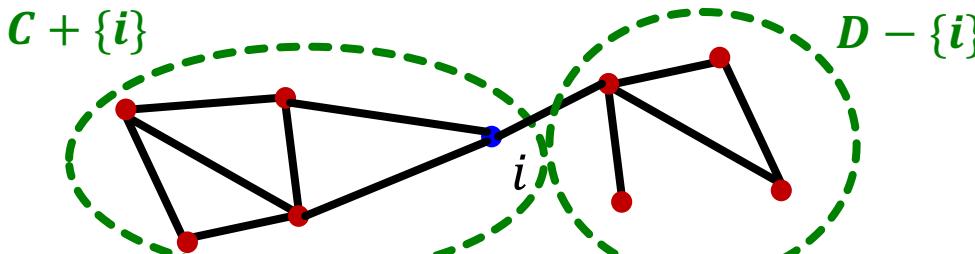
Intermediate:



Merging i into C

$$\Delta Q(i \rightarrow C)$$

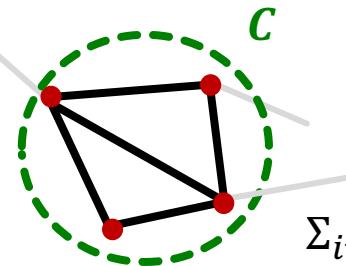
After:



Deriving $\Delta Q(i \rightarrow C)$

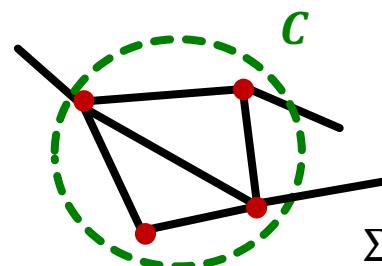
- Let's derive $\Delta Q(i \rightarrow C)$
 - First, we derive modularity **within** C , i.e., $Q(C)$.
 - Define:**
 - $\Sigma_{in} \equiv \sum_{i,j \in C} A_{ij}$... sum of link weights between nodes in C
 - $\Sigma_{tot} \equiv \sum_{i \in C} k_i$... sum of all link weights of nodes in C

Σ_{in} :



$$\Sigma_{in} = 10$$

Σ_{tot} :



$$\Sigma_{tot} = 13$$

Deriving $\Delta Q(i \rightarrow C)$

- Let's derive $\Delta Q(i \rightarrow C)$
- First, we derive modularity **within C** , i.e., $Q(C)$.
- Define:**
 - $\Sigma_{in} \equiv \sum_{i,j \in C} A_{ij}$... sum of link weights between nodes in C
 - $\Sigma_{tot} \equiv \sum_{i \in C} k_i$... sum of all link weights of nodes in C
- Then, we have

$$Q(C) \equiv \frac{1}{2m} \sum_{i,j \in C} \left[A_{ij} - \frac{k_i k_j}{2m} \right] = \frac{\Sigma_{i,j \in C} A_{ij}}{2m} - \frac{(\Sigma_{i \in C} k_i)(\Sigma_{j \in C} k_j)}{(2m)^2}$$
$$= \frac{\Sigma_{in}}{2m} - \frac{\left(\frac{\Sigma_{tot}}{2m} \right)^2}{2m}$$

Links within the community $\frac{\Sigma_{in}}{2m}$ - Total links $\left(\frac{\Sigma_{tot}}{2m} \right)^2$

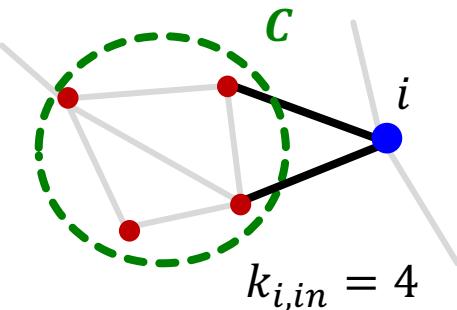
$Q(C)$ is large when most of the total links are within-community links

Deriving $\Delta Q(i \rightarrow C)$

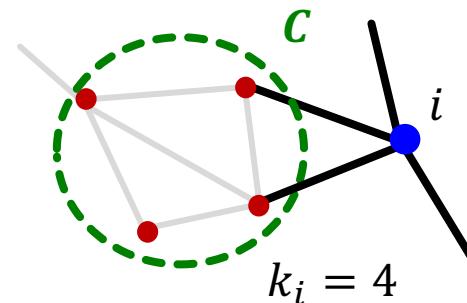
■ Further define:

- $k_{i,in} \equiv \sum_{j \in C} A_{ij} + \sum_{j \in C} A_{ji}$... sum of link weights between node i and C
- k_i ... sum of all link weights (i.e., degree) of node i

$k_{i,in} :$

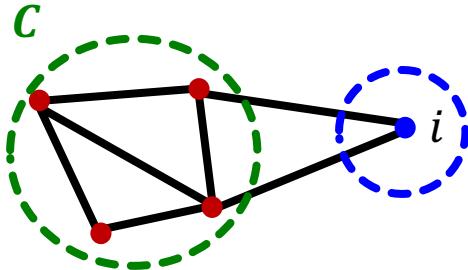


$k_i :$



Deriving $\Delta Q(i \rightarrow C)$

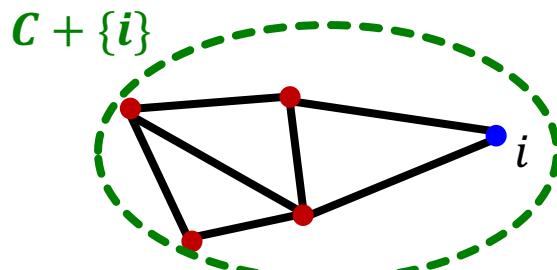
Before merging



Isolated community of node i

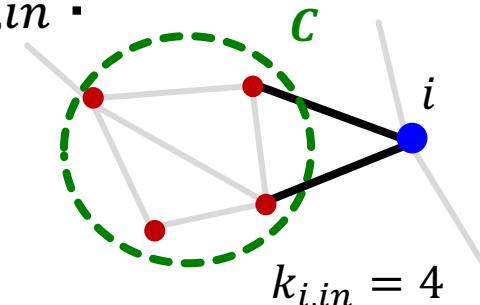
$$Q_{\text{before}} = Q(C) + Q(\{i\}) \\ = \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m} \right)^2 \right] + \left[0 - \left(\frac{k_i}{2m} \right)^2 \right]$$

After merging

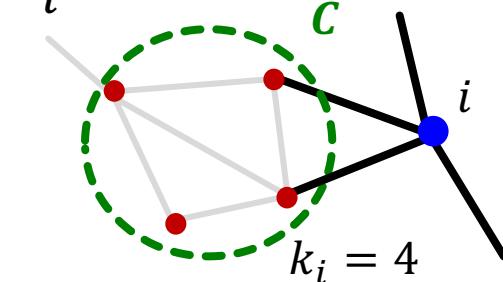


$$Q_{\text{after}} = Q(C + \{i\}) \\ \text{"}\Sigma_{in}\text{" of } C + \{i\} \quad \text{"}\Sigma_{tot}\text{" of } C + \{i\} \\ = \frac{\Sigma_{in} + k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m} \right)^2$$

Recall: $k_{i,in}$:



k_i :



Louvain: Modularity Gain

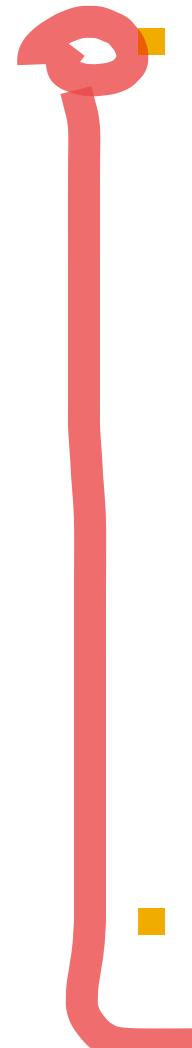
- $\Delta Q(i \rightarrow C) = Q_{\text{after}} - Q_{\text{before}}$
$$= \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m} \right)^2 \right]$$
$$- \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right]$$
- $\Delta Q(D \rightarrow i)$ can be derived similarly.
- **In summary, we can compute:**
$$\Delta Q(D \rightarrow i \rightarrow C) = \Delta Q(D \rightarrow i) + \Delta Q(i \rightarrow C)$$

Louvain 1st Phase: Summary

- Iterate until no node moves to a new community:

- 
- For each node $i \in V$ currently in community C , compute the **best community C'** :
 - $C' = \operatorname{argmax}_{C'} \Delta Q(C \rightarrow i \rightarrow C')$
 - If $\Delta Q(C \rightarrow i \rightarrow C') > 0$, then **update the community**:
 - $C \leftarrow C - \{i\}$
 - $C' \leftarrow C' + \{i\}$

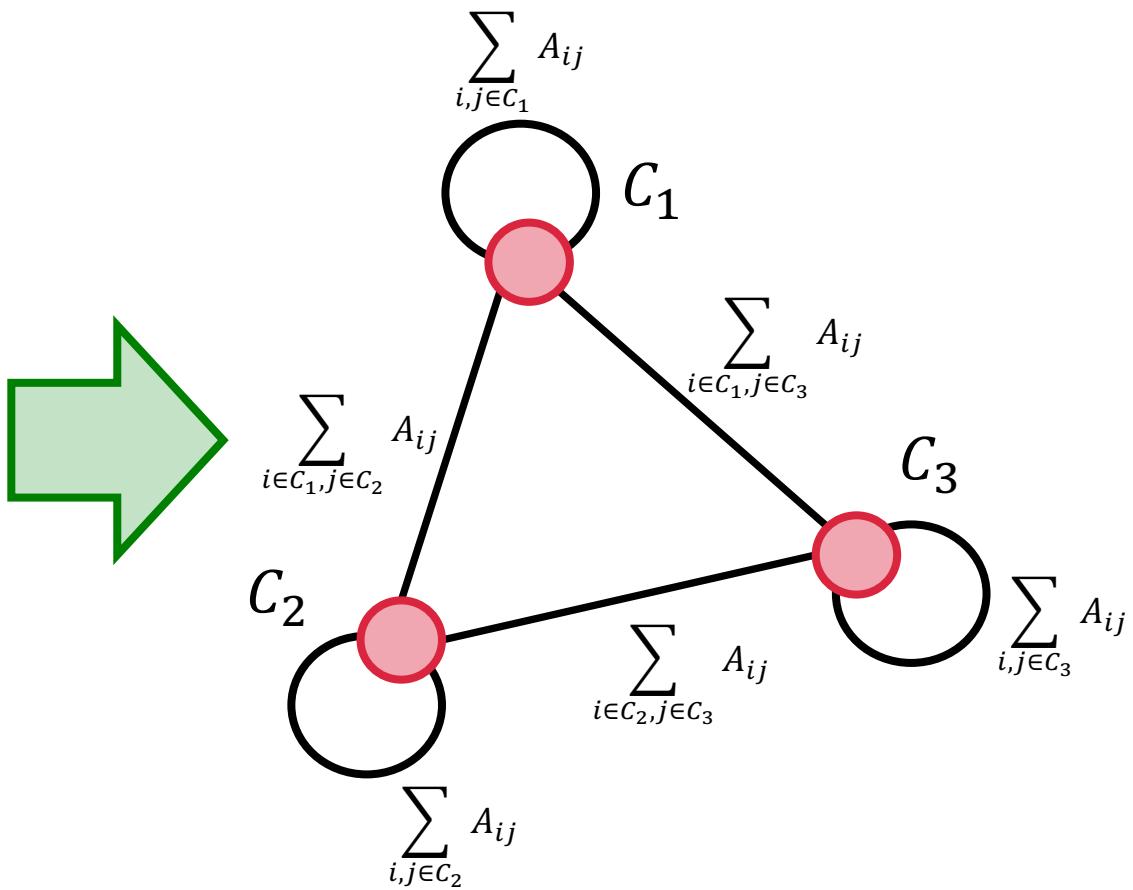
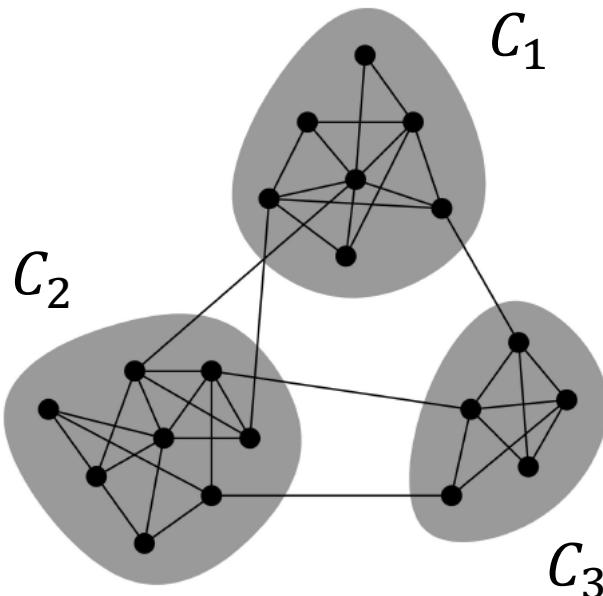
Louvain: 2nd phase (Restructuring)

- 
- The communities obtained in the first phase are contracted into **super-nodes**, and the network is created accordingly:
 - Super-nodes are connected if there is at least one edge between the nodes of the corresponding communities
 - The weight of the edge between the two super-nodes is the sum of the weights from all edges between their corresponding communities
 - Phase 1 is then run on the super-node network**

Louvain 2st Phase: Summary

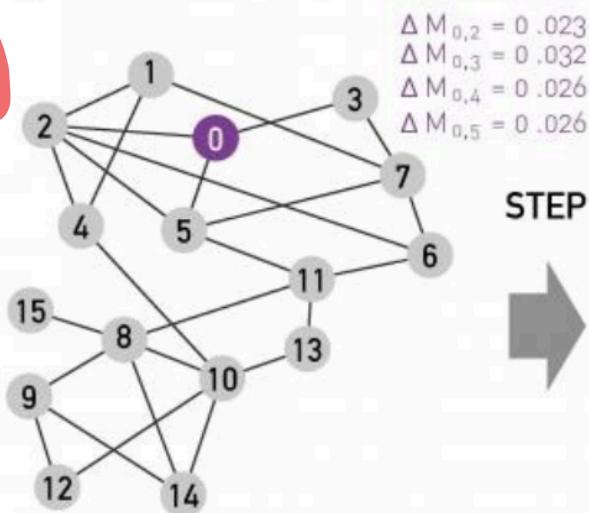
- Super nodes are constructed by merging nodes in the same community.

Community assignment
obtained after 1st phase

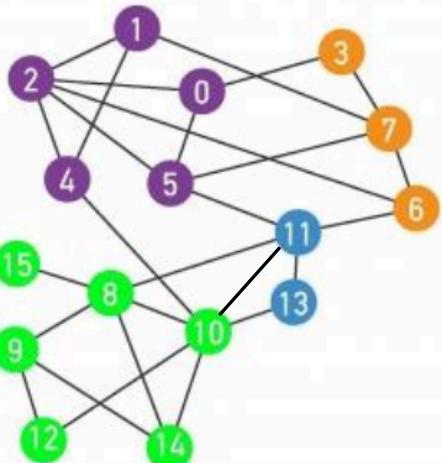


Louvain Algorithm

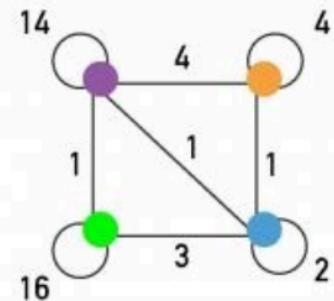
1ST PASS



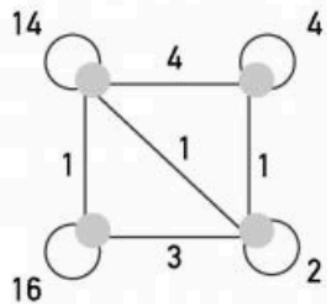
STEP I



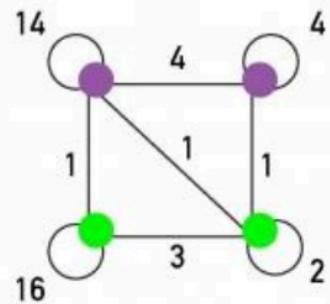
STEP II



2ND PASS



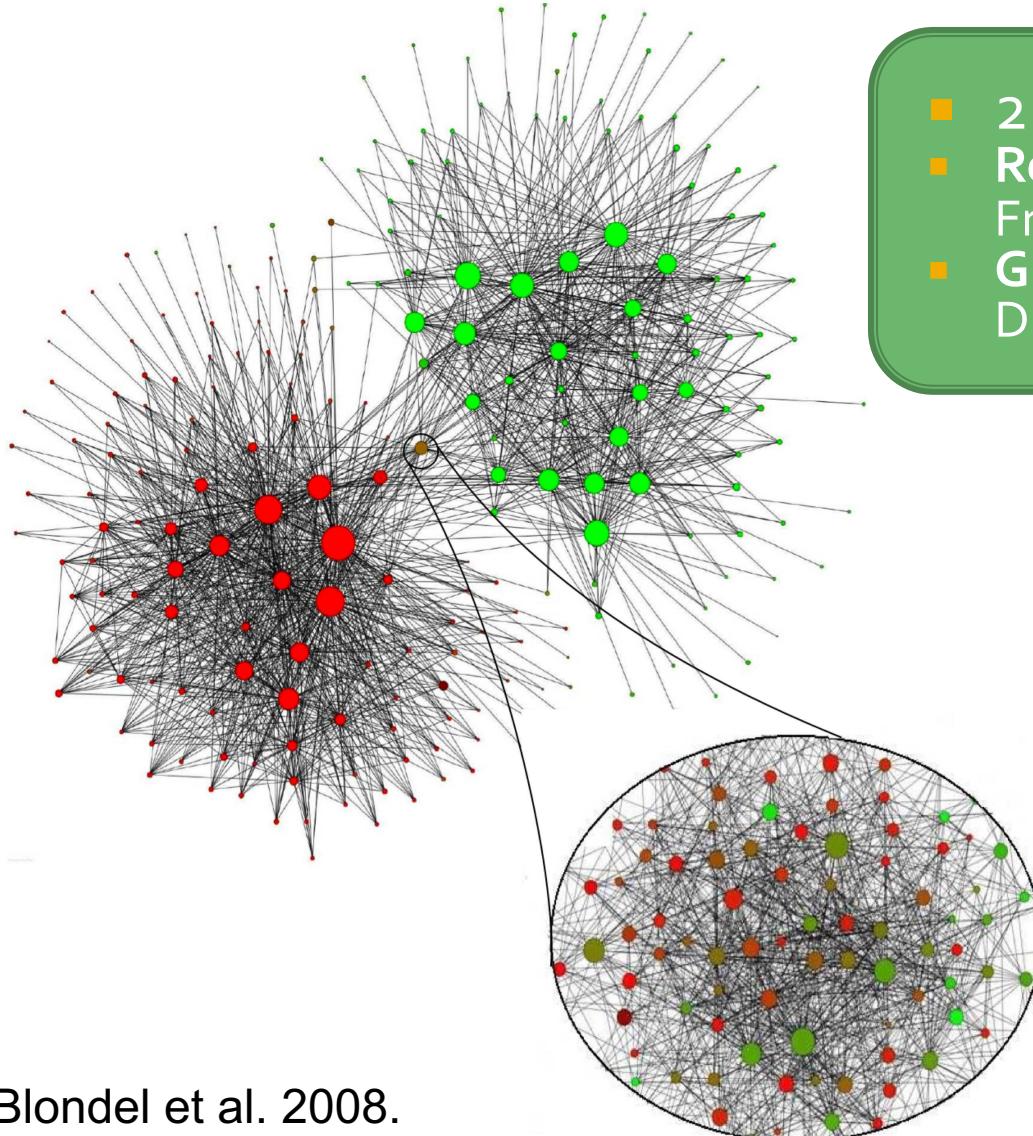
STEP I



STEP II



Belgian Mobile phone network



Adopted from Blondel et al. 2008.

Summary: Modularity

- **Modularity:**
 - Overall quality of the partitioning of a graph into communities
 - Used to determine the number of communities
- **Louvain modularity maximization:**
 - Greedy strategy
 - Great performance, scales to large networks

Stanford CS224W: Detecting Overlapping Communities: BigCLAM

CS224W: Machine Learning with Graphs

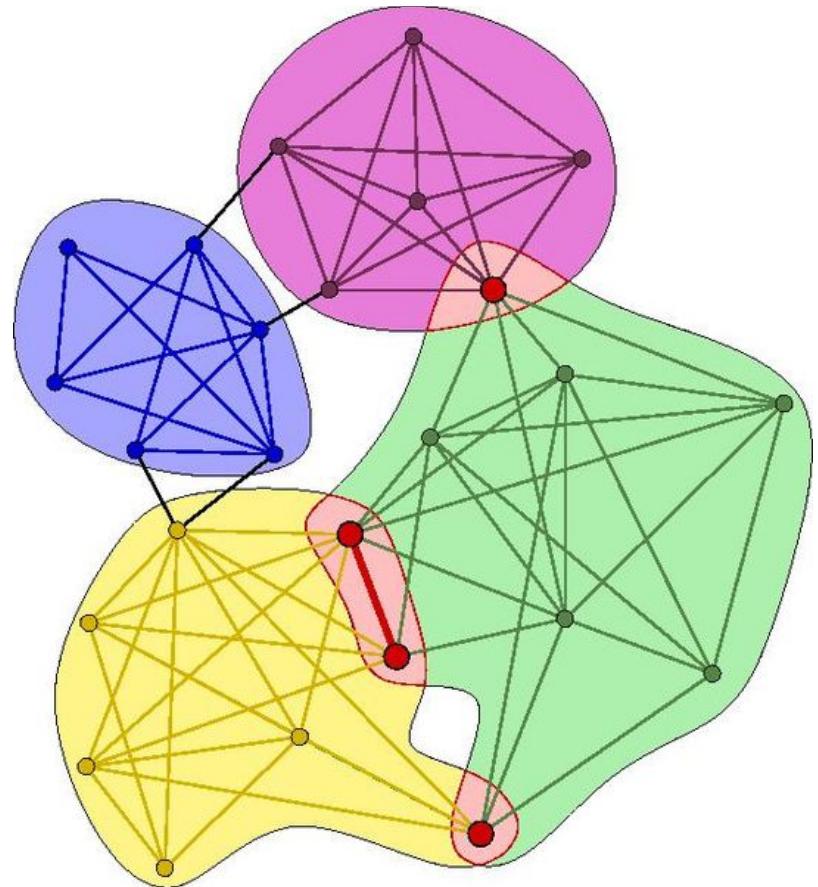
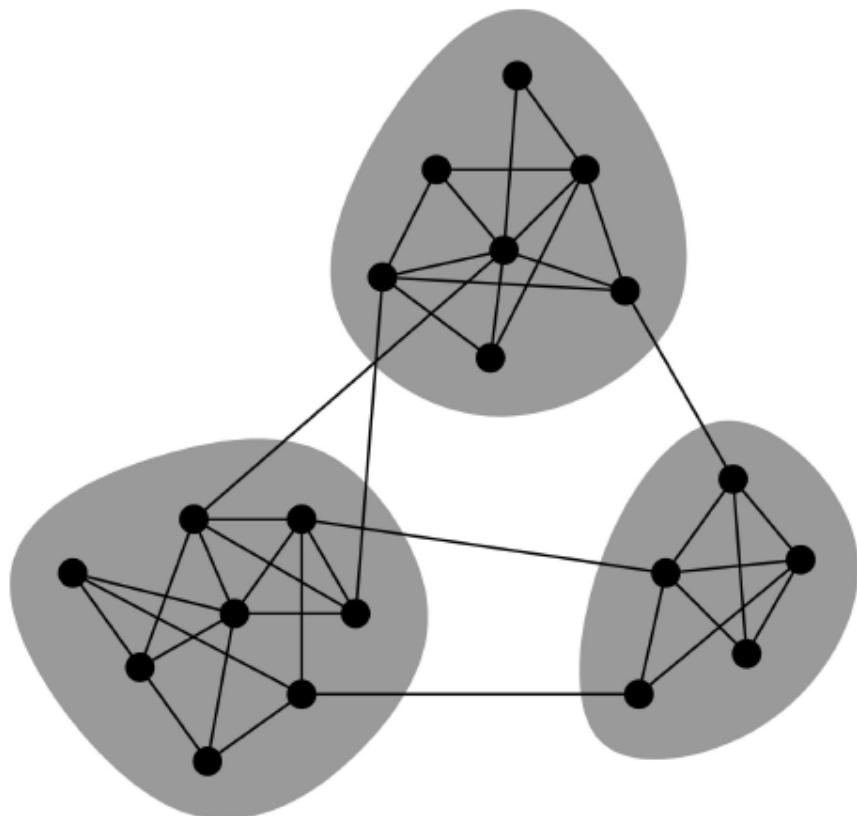
Jure Leskovec, Stanford University

<http://cs224w.stanford.edu>

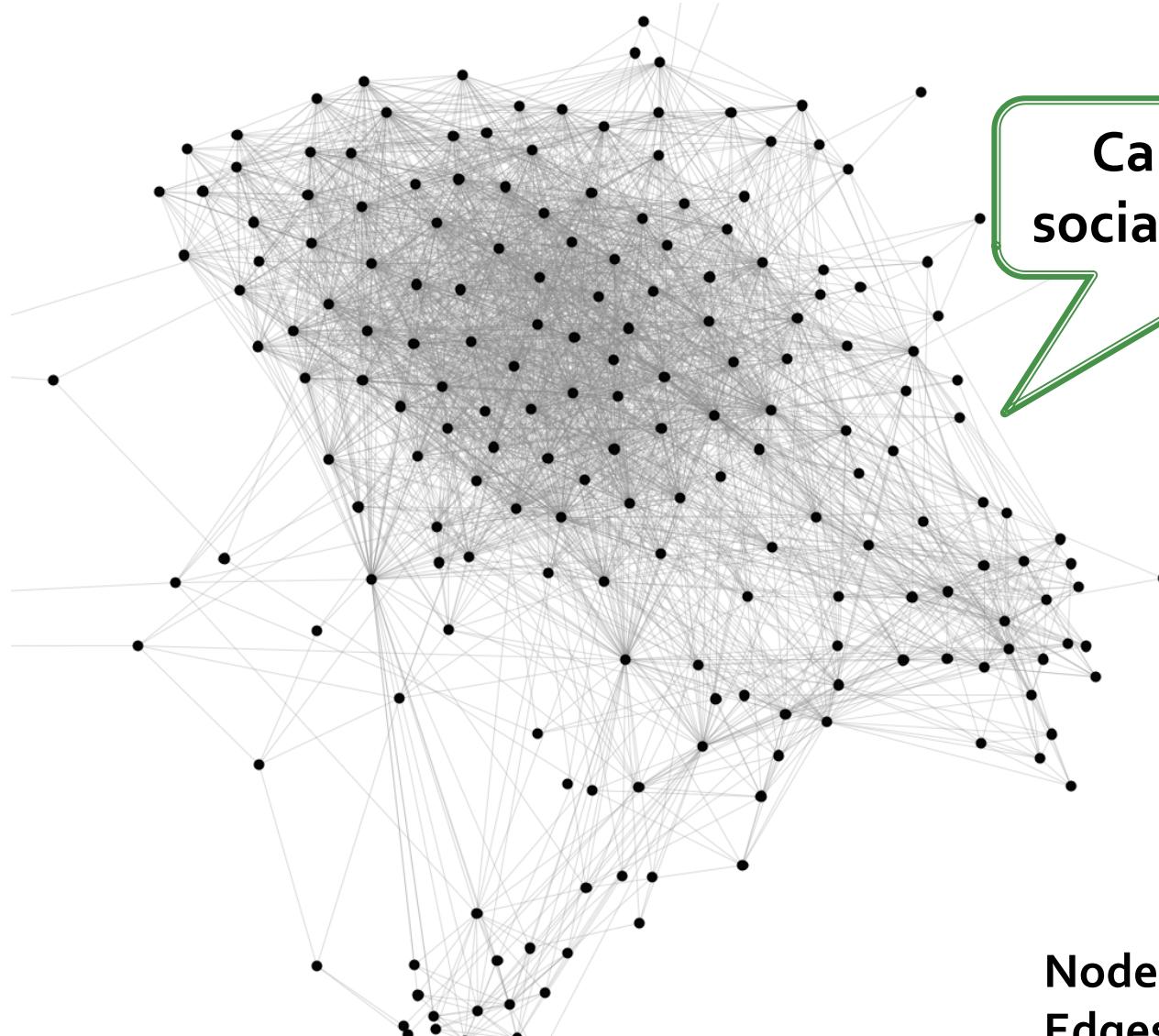


Overlapping Communities

- Non-overlapping vs. overlapping communities

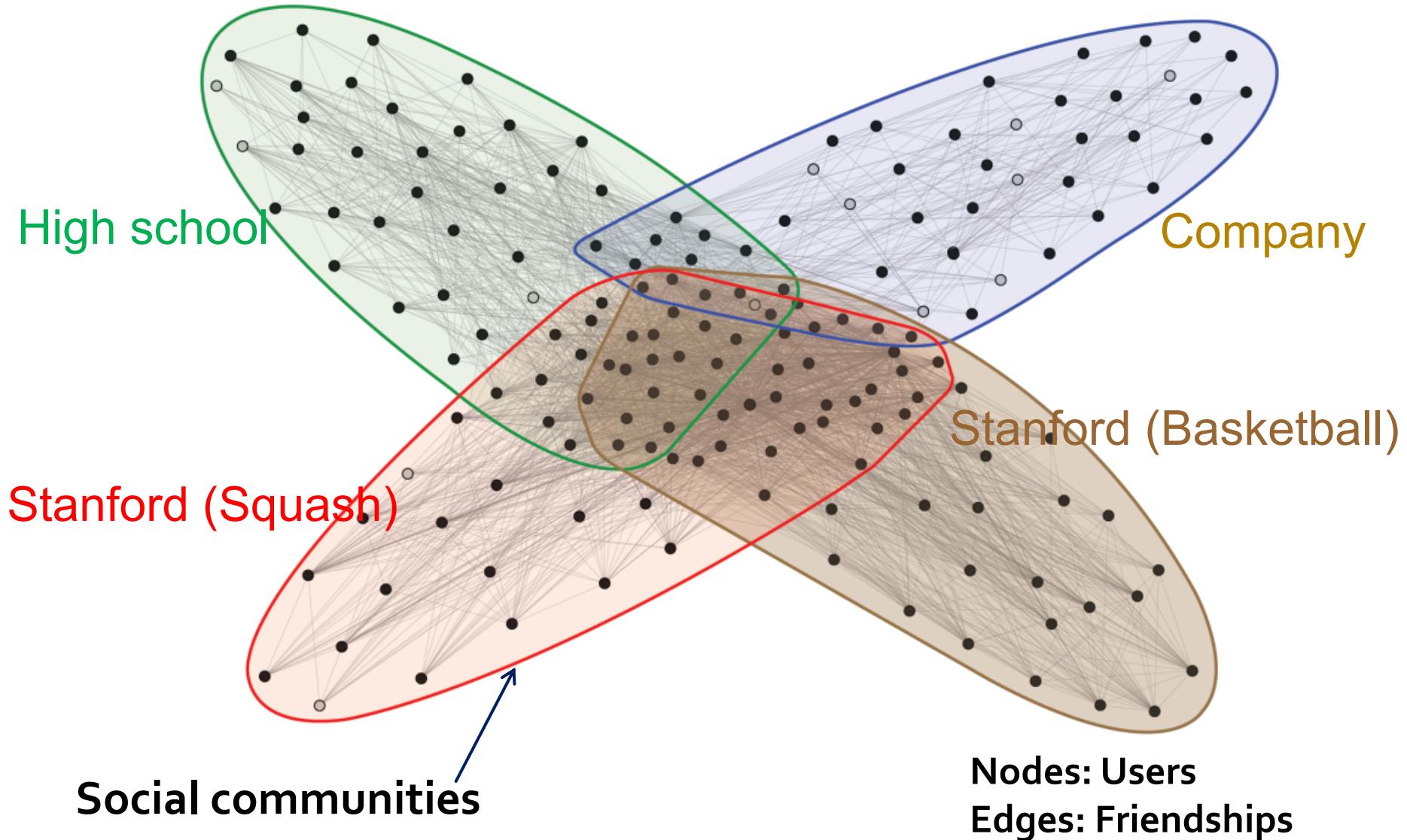


Facebook Ego-network

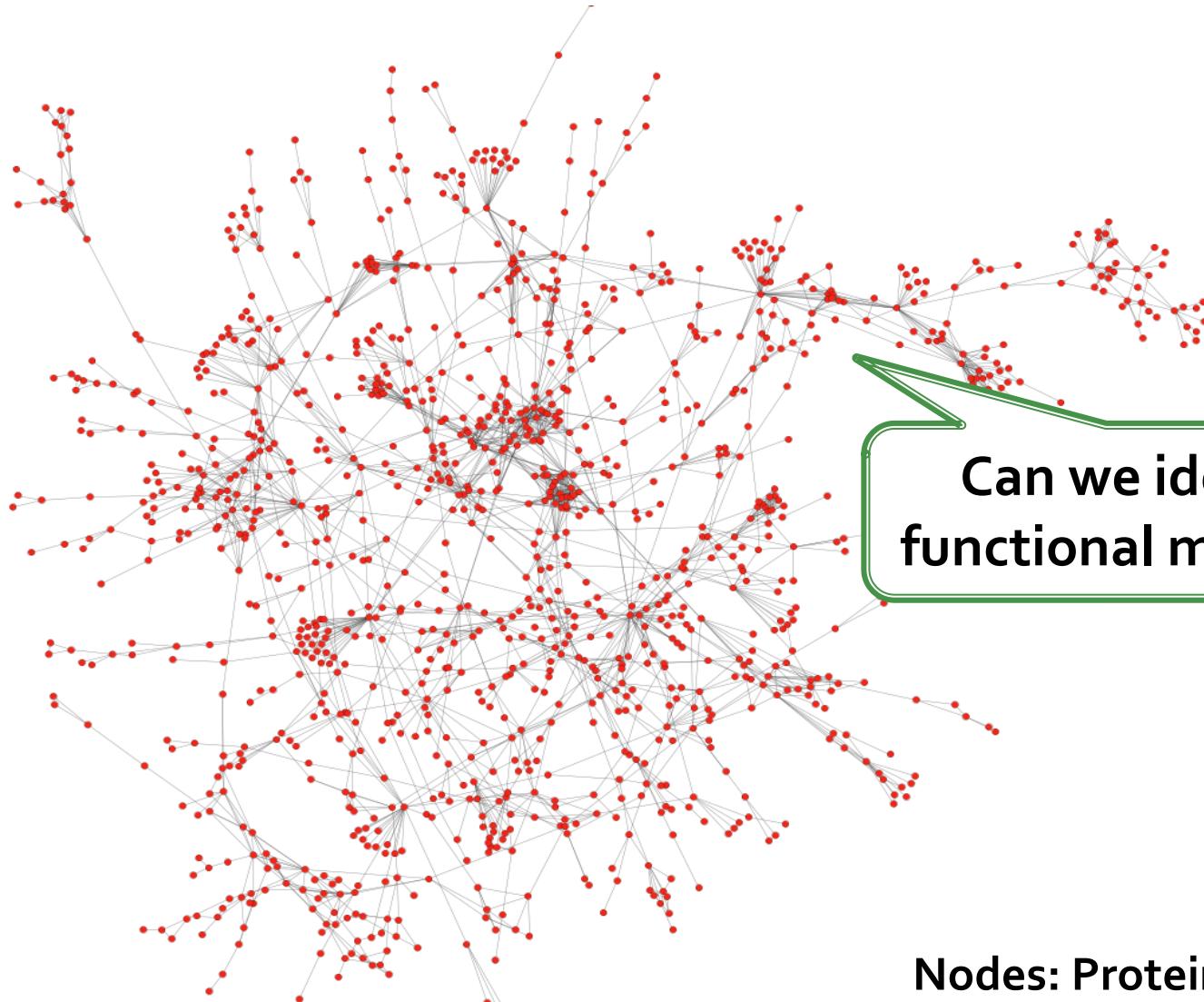


**Nodes: Users
Edges: Friendships**

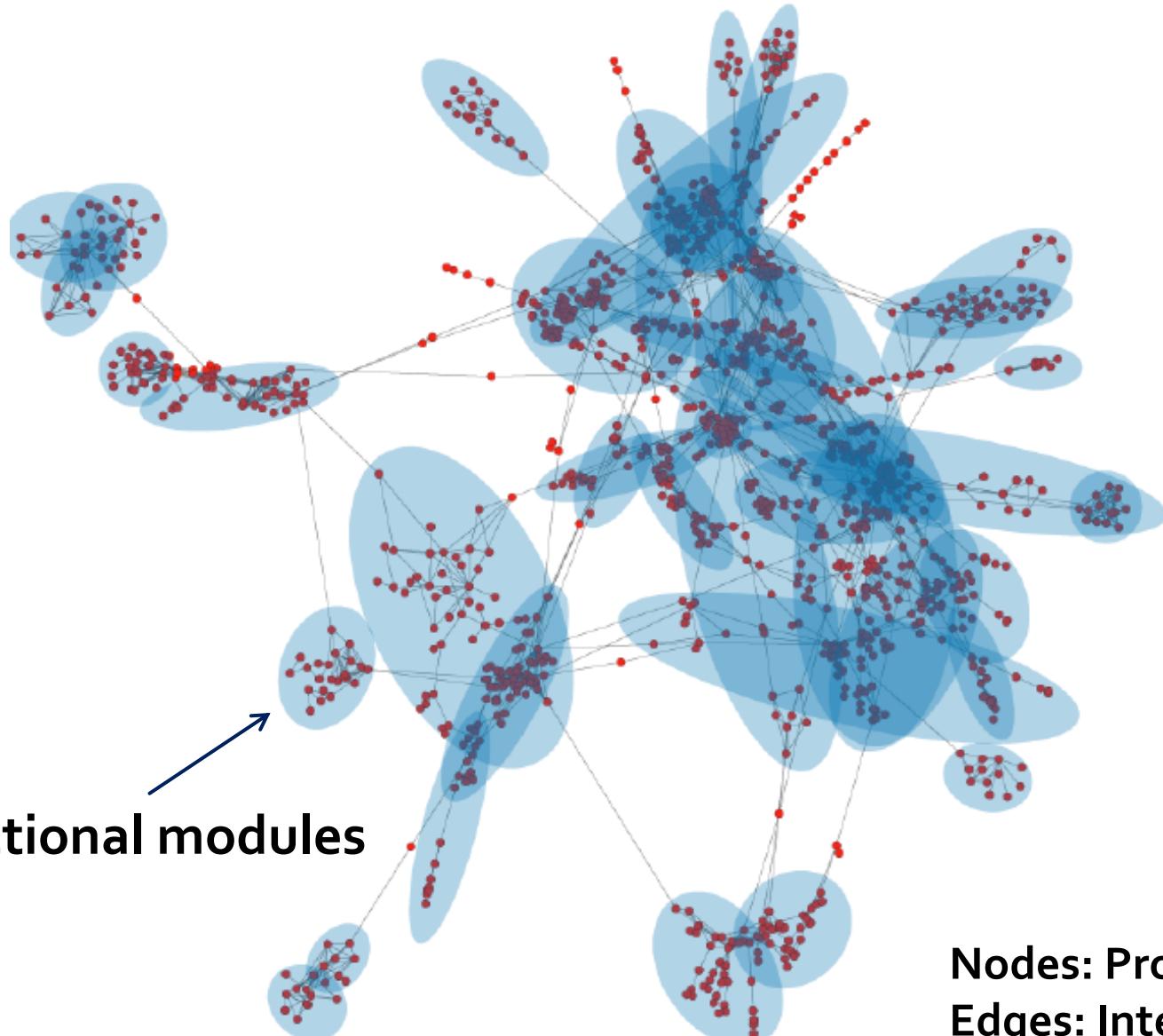
Facebook Ego-network



Protein-Protein Interactions

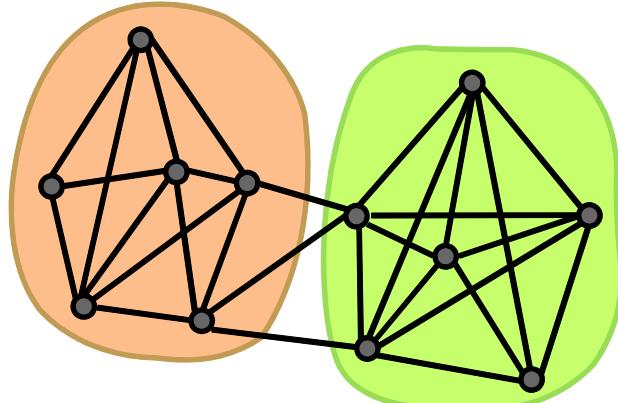


Protein-Protein Interactions

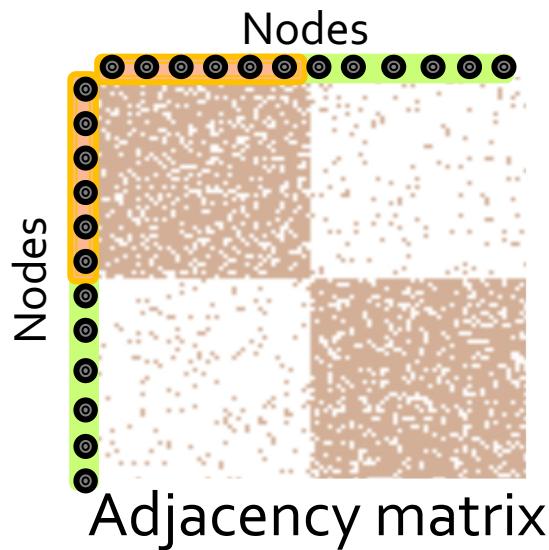


Communities

Non-overlapping

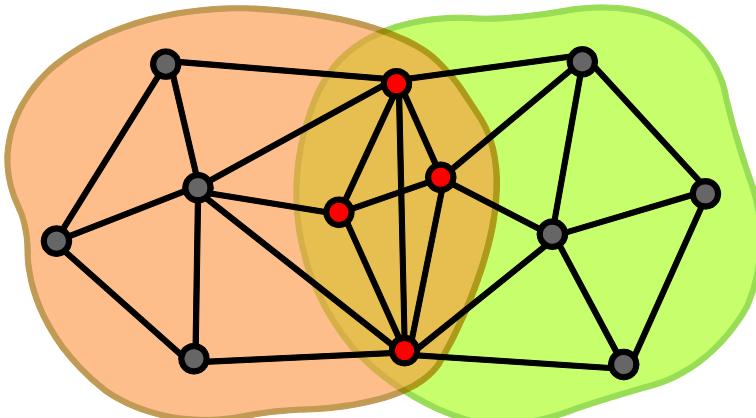


Network



Adjacency matrix

Overlapping



Plan of Action

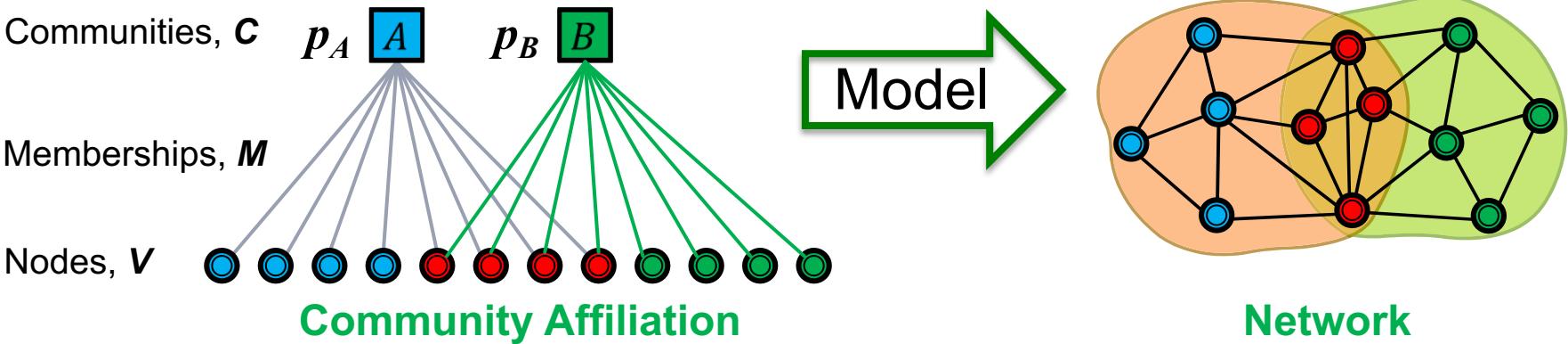
Step 1)

- Define a generative model for graphs that is based on node community affiliations
 - **Community Affiliation Graph Model (AGM)**

Step 2)

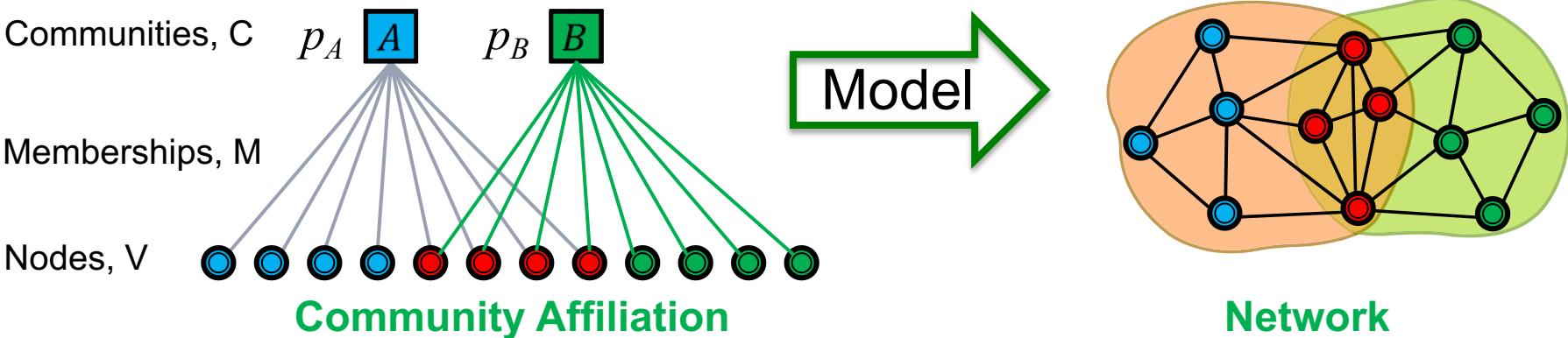
- Given graph G , make the assumption that G was generated by AGM
- Find the best AGM that could have generated G
- **And this way we discover communities**

Community-Affiliation Graph Model (AGM)



- **Generative model:** How is a network generated from community affiliations?
- **Model parameters:**
 - Nodes V , Communities C , Memberships M
 - Each community c has a single probability p_c

AGM: Generative Process

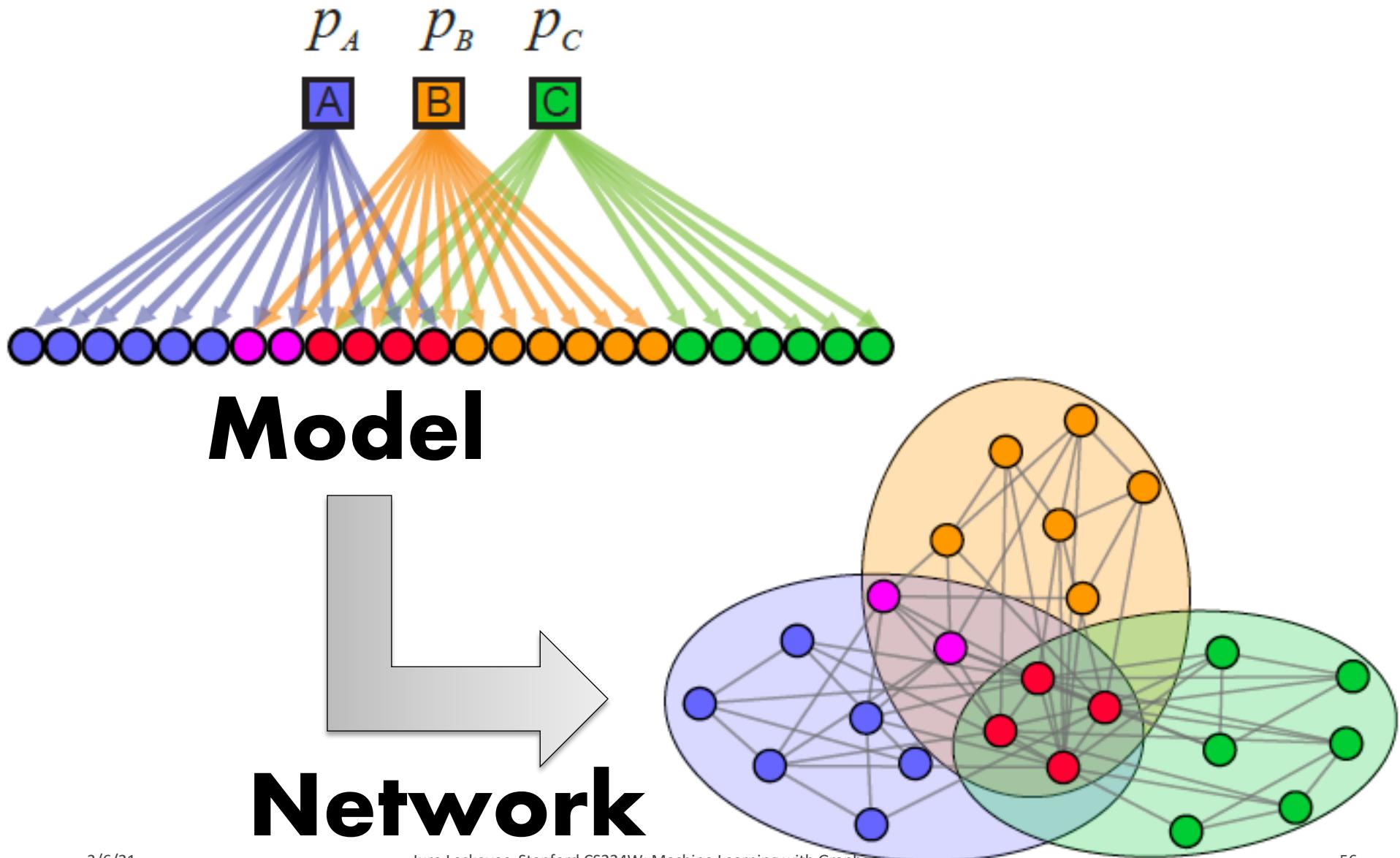


- Given parameters ($V, C, M, \{p_c\}$)
 - Nodes in community c connect to each other by flipping a coin with probability p_c
 - **Nodes that belong to multiple communities have multiple coin flips**
 - If they “miss” the first time, they get another chance through the next community

$$p(u, v) = 1 - \prod_{c \in M_u \cap M_v} (1 - p_c)$$

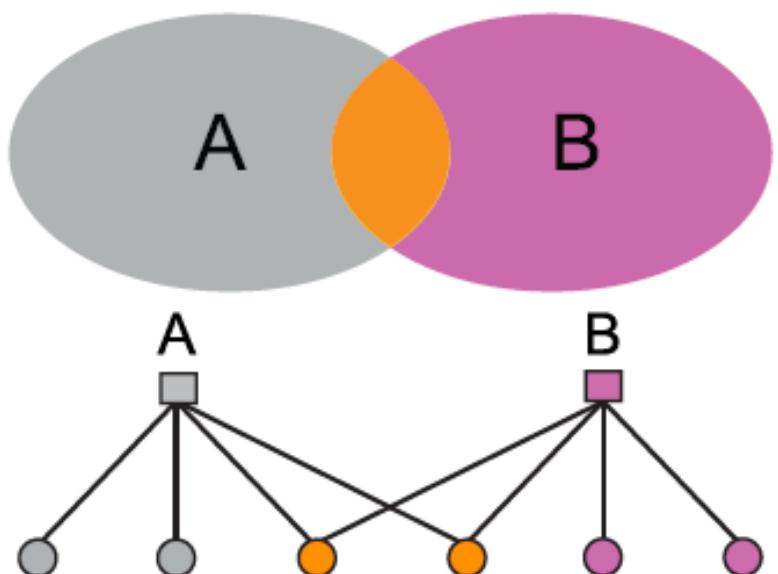
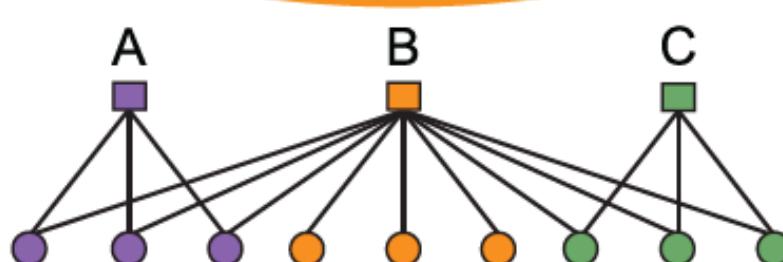
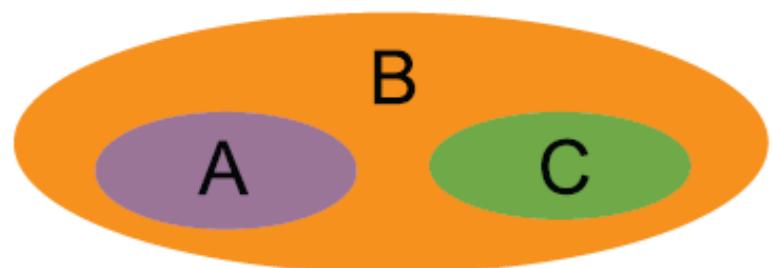
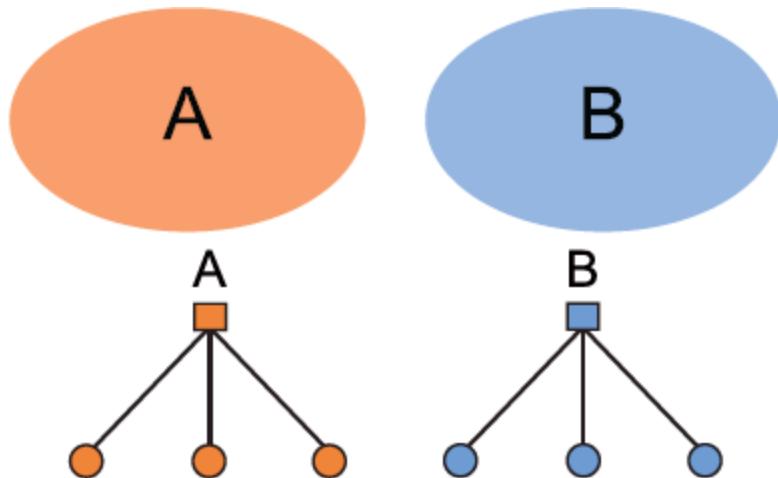
Note: If nodes u and v have no communities in common, then $p(u,v)=0$. We resolve this by having a background “epsilon” community that every node is a member of.

AGM: Dense Overlaps



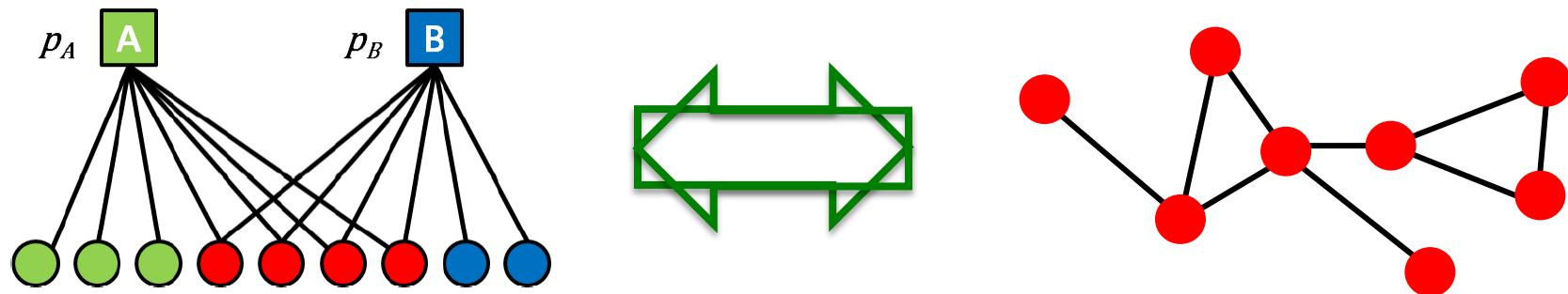
AGM: Flexibility

- AGM can express a variety of community structures:
Non-overlapping,
Overlapping, Nested



Detecting Communities

- Detecting communities with AGM:



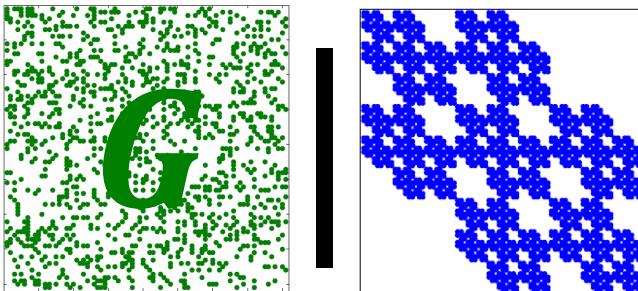
Given a Graph, find the model F

- 1) Affiliation graph M
- 2) Number of communities C
- 3) Parameters p_c

Graph Fitting

How to estimate model parameters F given a G ?

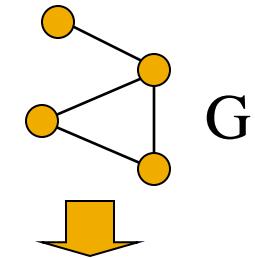
- Maximum likelihood estimation
- Given real graph G
- Find model/parameters F which

$$\arg \max_F P(G | F)$$


- To solve this we need to:
 - Efficiently calculate $P(G|F)$
 - Then maximize over F (e.g., using gradient descent)

Graph Likelihood $P(G|F)$

- Given G and F we calculate likelihood that F generated G : $P(G|F)$



F

0.25	0.10	0.10	0.04
0.05	0.15	0.02	0.06
0.05	0.02	0.15	0.06
0.01	0.03	0.03	0.09

$P(u, v)$: Edge prob. of edge (u, v)

1	0	1	1
0	1	0	1
1	0	1	1
1	1	1	1

$P(G|F)$

G

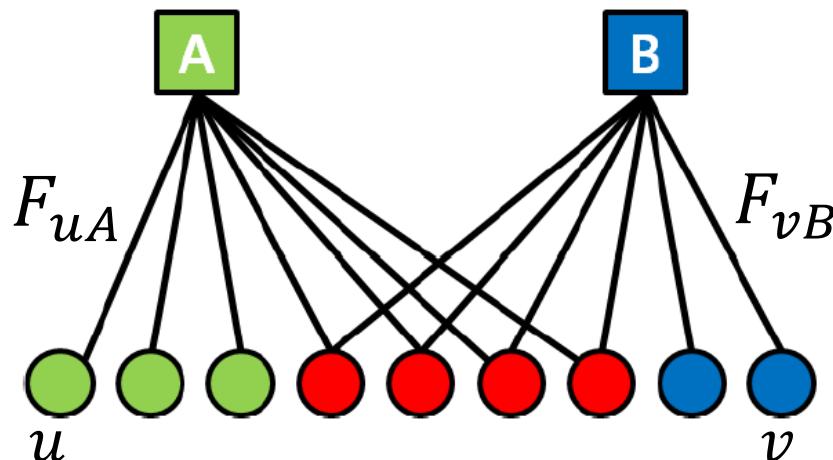
$$P(G|F) = \prod_{(u,v) \in G} P(u, v) \prod_{(u,v) \notin G} (1 - P(u, v))$$

Likelihood of edges in the graph

Likelihood of edges not in the graph

“Relaxing” AGM: Towards $P(u, v)$

- “Relax” the AGM: Memberships have strengths



- F_{uA} : The membership strength of node u to community A ($F_{uA} = 0$: no membership)

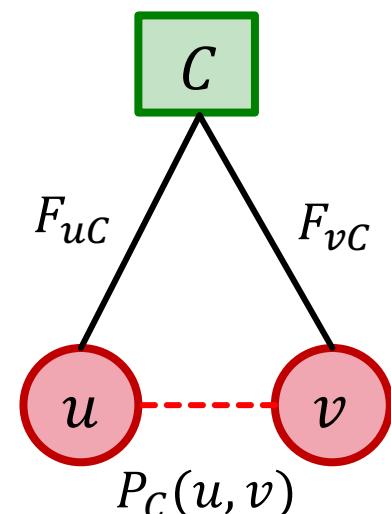
“Relaxing” AGM: Towards $P(u, v)$

- For community C , we model the probability of u and v being connected as

$$P_C(u, v) = 1 - \exp(-F_{uC} \cdot F_{vC})$$

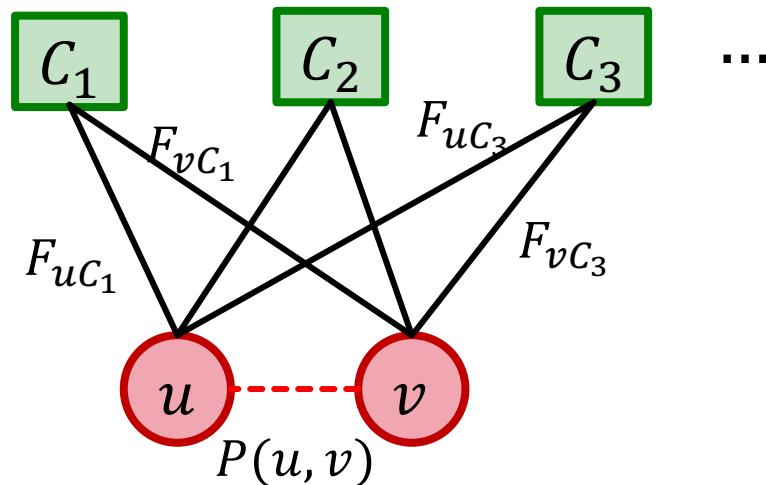
Non-negative
membership
strength

- $P_C(u, v)$ satisfies $0 \leq P_C(u, v) \leq 1$ (**valid probability**) because $F_{uC} \cdot F_{vC} \geq 0$.
 - $P_C(u, v) = 0$ iff $F_{uC} \cdot F_{vC} = 0$ (i.e., $F_{uC} = 0$ or $F_{vC} = 0$)
 - Nodes u or v are **not** connected via C iff **at least one of them** has zero membership strength for C .
 - $P_C(u, v) \approx 1$ iff $F_{uC} \cdot F_{vC}$ is large.
 - Nodes u or v are connected via C iff **both** u and v have **high** membership strength for C .



“Relaxing” AGM: Towards $P(u, v)$

- Nodes u and v can be connected via multiple communities $\in \Gamma$ (a set of all communities).



- Probability that u and v are connected by at least one of the communities:

$$P(u, v) = 1 - \prod_{C \in \Gamma} (1 - P_C(u, v))$$

Probability that u and v are *not* connected by any communities

“Relaxing” AGM: Towards $P(u, v)$

- Expanding $P(u, v)$:

$$\begin{aligned} P(u, v) &= 1 - \prod_{C \in \Gamma} (1 - P_C(u, v)) \\ &\quad 1 - \exp(-F_{uC} \cdot F_{vC}) \\ &= 1 - \prod_{C \in \Gamma} \exp(-F_{uC} \cdot F_{vC}) \\ &= 1 - \exp\left(-\sum_{c \in \Gamma} F_{uC} \cdot F_{vC}\right) \\ &= 1 - \exp(-\underbrace{\mathbf{F}_u^T \mathbf{F}_v}_{\text{Inner product}}) \end{aligned}$$

\mathbf{F}_{uC} : A vector of $\{F_{uC}\}_{C \in \Gamma}$
 \mathbf{F}_{vC} : A vector of $\{F_{vC}\}_{C \in \Gamma}$

BigCLAM Model

- Prob. of nodes u, v linking is proportional to the strength of shared memberships:

$$P(u, v) = 1 - \exp(-F_u^T F_v)$$

- Given a network $G(V, E)$, we maximize the likelihood (probability) of G under our model

$$\begin{aligned} P(G|F) &= \prod_{(u,v) \in E} P(u, v) \prod_{(u,v) \notin E} (1 - P(u, v)) \\ &= \prod_{(u,v) \in E} (1 - \exp(-F_u^T F_v)) \prod_{(u,v) \notin E} \exp(-F_u^T F_v) \end{aligned}$$

BigCLAM Model

- Likelihood involves a product of many small probabilities → Numerically unstable.
- We consider the **log** likelihood:
 $\log(P(G|F))$

$$= \log \left(\prod_{(u,v) \in E} (1 - \exp(-F_u^T F_v)) \prod_{(u,v) \notin E} \exp(-F_u^T F_v) \right)$$

$$= \sum_{(u,v) \in E} \log(1 - \exp(-F_u^T F_v)) - \sum_{(u,v) \notin E} F_u^T F_v$$

$\equiv \ell(F)$: Our objective

BigCLAM Model

- **Optimizing $\ell(\mathbf{F})$**
- Start with random membership \mathbf{F}
- Iterate until convergence
 - For $u \in V$
 - Update membership \mathbf{F}_u for node u while fixing the memberships of all other nodes
 - Specifically, we do **gradient ascent**, where we make small changes to \mathbf{F}_u that lead to increase in log-likelihood.

Partial derivative w.r.t. \mathbf{F}_u

$$\nabla \ell(\mathbf{F}) = \sum_{v \in \mathcal{N}(u)} \left(\frac{\exp(-\mathbf{F}_u^T \mathbf{F}_v)}{1 - \exp(-\mathbf{F}_u^T \mathbf{F}_v)} \right) \cdot \mathbf{F}_v - \sum_{v \notin \mathcal{N}(u)} \mathbf{F}_v$$

BigCLAM Model

- **Time complexity of gradient ascent:**

Linear in the degree of u (fast)

Linear in #nodes (slow)

$$\nabla \ell(\mathbf{F}) = \sum_{v \in \mathcal{N}(u)} \left(\frac{\exp(-\mathbf{F}_u^T \mathbf{F}_v)}{1 - \exp(-\mathbf{F}_u^T \mathbf{F}_v)} \right) \cdot \mathbf{F}_v - \sum_{v \notin \mathcal{N}(u)} \mathbf{F}_v$$

- Naïve gradient ascent is slow! **However:**

Can be efficiently updated!

$$\sum_{v \notin \mathcal{N}(u)} \mathbf{F}_v = \sum_v \mathbf{F}_v - \mathbf{F}_u - \sum_{v \in \mathcal{N}(u)} \mathbf{F}_v$$

- $\sum_v \mathbf{F}_v$ can be computed at the beginning and efficiently updated whenever \mathbf{F}_u is updated.
- The gradient step takes **linear time** in the degree of u !

BigCLAM: Summary

- BigCLAM defines the model to generate a network with **overlapping community structure**.
- Given a graph, BigCLAM's parameters (**membership strength of each node**) can be estimated by maximizing the log-likelihood of generating the graph under the model.

